CS & IT ENGINEERING

Computer Organization and Architecture

Secondary Memory & IO Interface

Lecture No.- 06

By- Vijay Agarwal Sir

Recap of Previous Lecture

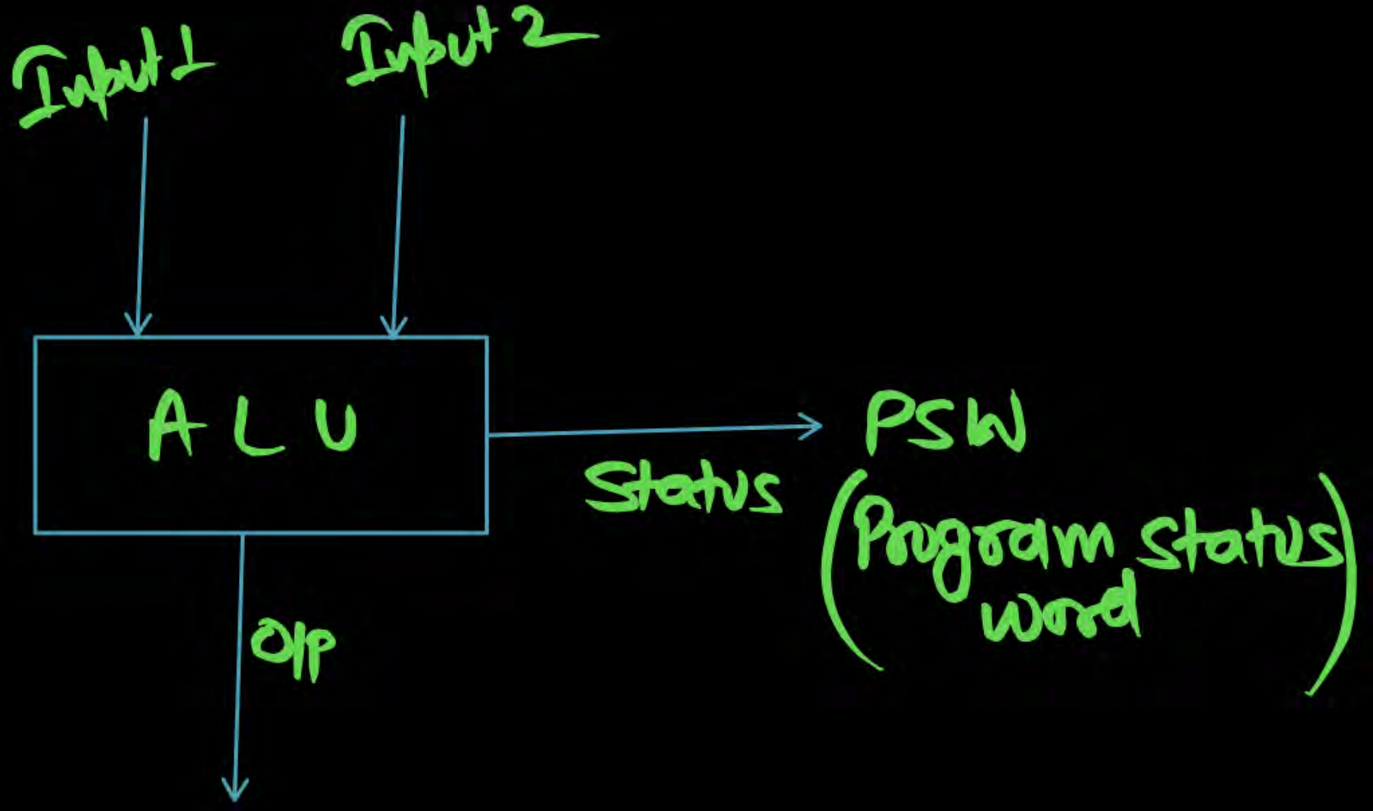
- Topic
- Topic

IO Organization

Topics to be Covered

- Topic: Flag
- Topic: Shift & Rotate Operation
- Topic: Interrupt Cycle
- Topic: Amdhal's Law & Flynn Classification
- Topic: System Bus
- Topic: IO Interface & DMA

Flag :



Flags

- **PSW (Program Status Word) : PSW contain the accumulator content along with Flag Register to hold the status of a instruction during the execution.**

[PSW = Accumulator & Flag Register pair]

1. Flag is a flip-flop, it is a bi-state component, used to store one-bit in formation.
2. Flag's are categorized into two groups:
   (1) Conditional Flag
   (2) Control Flag

Flags

Conditional Flag:

These flags are set and reset based on the result nature of ALU, with reference to a 8086 flag register, 6 conditional flag's are present.

Flag's are present namely as:

1. Carry flag
2. Parity Flag
3. Auxiliary carry flag
4. Zero flag
5. Sign Flag
6. Overflow Flag

1. Carry Flags

(1) **Carry:** Condition is "Is there an extra bit out " of MSB

True → T = Set = 1 = C (Carry)

False → F = reset = 0 = NC (Not Carry)

1. Carry Flags

Carry Flag: **Carry Flag is used in the unsigned arithmetic to** indicate the range exceeding condition.

n bit unsigned Range is : $0 \text{ to } 2^n - 1$

4 bit unsigned Range is : $0 \text{ to } 2^4 - 1 \Rightarrow (0 \text{ to } 15)$

1. Carry Flags

Eg.

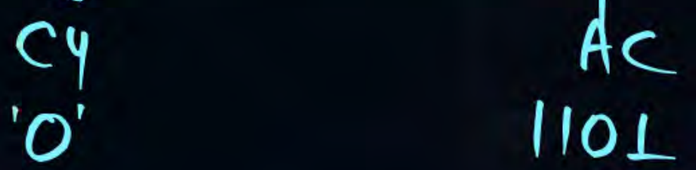
```
  6
+ 7
-----
 13   (in the Range)
```

Carry = 0 = Reset

$C_y = 0$

```
  0 1 1 0
+ 0 1 1 1
---------
  1 1 0 1   (13)
```

PSW

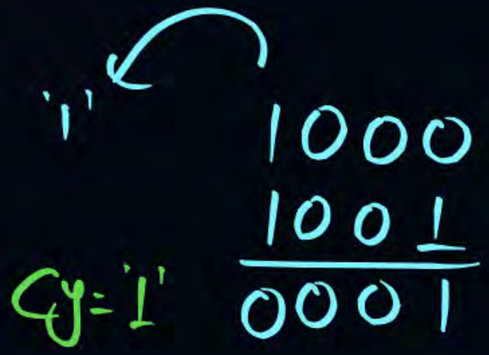


1. Carry Flags

Eg.

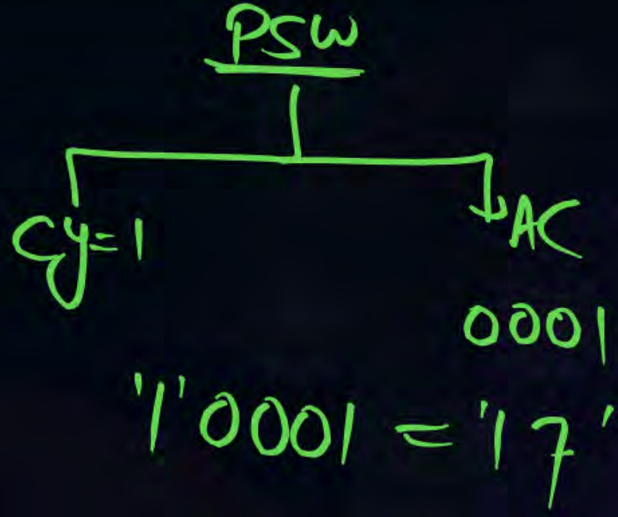
$$\begin{array}{r} 8 \\ +\,9 \\ \hline 17 \end{array}$$

But Range is (0 to 15)



Carry Flag Set

$Cy = 1$



UNSIGNED DATA

"Is three an extra bit out of MSB"

- T = Set = 1 = carry
- F = Reset = 0 = No carry

Ex-

6	–	0 1 1 0
+7	–	0 1 1 1
13		1 0 0 1
Cy =0		Carry = 0

Justify: PSW

Accumulator: 1 1 0 1

1101 = [13]

Flog Reg. cy = 0

EX-

8	–	1 0 0 0
+9	–	1 0 0 1
17		0 0 0 1
cy =1		cy =1

Justify: PSW

ACC: 0 0 0 1

CY → 10001 = 17 (17)

Flog Reg. cy = 1

2. Parity Flags

(2) **Parity:** Condition is "The ALU O/P (ACC) contain even number of 1's"-

- T = Set = 1 = PE (Parity Even)
- F = result = 0 = PO (Parity ODD)

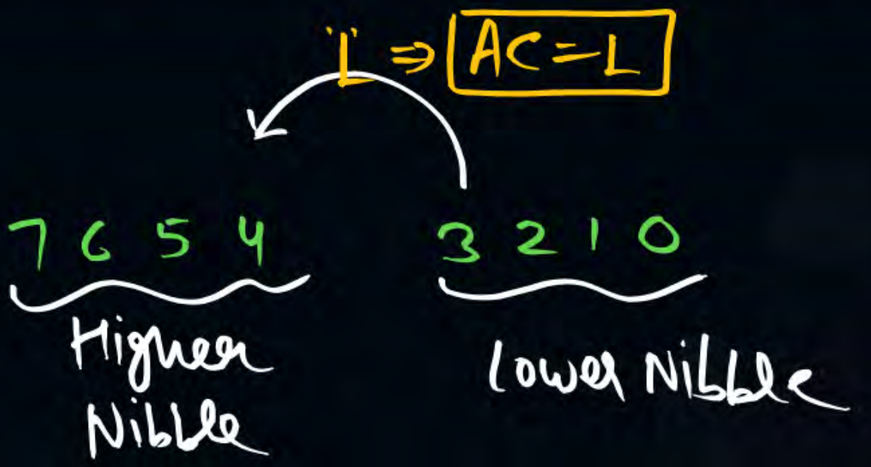
This flag is used in the serial data communication to prepare the error correction and error detection codes.

3. Auxiliary Carry Flags

(3) Auxiliary Carry Flag:

Condition is "Is there an extra bit from the lower nibble to higher nibble".

- T = Set = 1 = C
- F = reset = 0 = NC



Nibble = 4bit

3. Auxiliary Carry Flags

- This flag is used in the **BCD Arithmetic to indicate the range exceeding condition of the lower nibble.**
  It is set when the lower nibble is greater than "F".
  Carry flag is also used in the BCD arithmetic to indicate the range exceeding condition of the higher nibble. It is set when the higher nibble is greater than 'F'.]

2. BCD format is used to represent the decimal data. It is two kinds:
   1. Unpacked BCD
   2. Packed BCD

3. Auxiliary Carry Flags

3. In an unpacked BCD format, each digit represented with 8-bit.

Eg. 23: 00000010 00000011

2 3

4. In a packed BCD format each digit represented with 4-bit.

0000 → 0

1001 → 9

10 to 15 → '6' Pattern are unused.

Eg.

23: 0010 0011

2 3

In this format 6 codes [A to F] **(1010[A] to 1111[F])** are unused. Therefore adjustment is required to correct the data i.e. Add constant '6' to nibble when the nibble is greater than '9' [nibble > 9].

Auxiliary Carry Flags

Eg1-

27	–	0010	0111
+26	–	0010	0110
53		0100	1101 [4DH]
		4	D

AC =0 Carry = 0

LN(13)< F, AC=0;
HN(5) <F, CY=0:

(D>9 & AC=0 So, Actual LN is D only : ADD '6' to LN.)
(4<9 & CY=0 So, Actual HN is 4 only : No Need to Adjust.)

CY=0

AC=0

0010 0111
0010 0110
0100 1101
4 D(13)

0100 1101
0110
0101 0011
(53) Ans 4D +6 =53

Auxiliary Carry Flags

Eg2-	28	–	0010	1000	
	+29	–	0010	1001	
	57		0101	0001	**(51H)**
			5	1	

AC =1 Carry = 0

LN(17)> F, **AC=1;**
HN(5) <F, **CY=0:**

(1<9 & AC=1 So, Actual LN is >F : ADD '6' to LN.)
(5<9 & CY=0 So, Actual HN is 5 only : No Need to Adjust.)

51 +6 =57

cy=0 '0' '1' AC=1

0010 1000
0010 1001
0101 0001
5 1

AC=1
Cy=0
(51)

5 1
+6
57

10001 = '17'

Auxiliary Carry Flags

Eg3-

89	–	1000	1001
+98	–	1001	1000
187		0010	0001 (21H)

AC =1 Carry = 1

LN(17)> F, **AC=1;**
HN(18)>F, **CY=1:**

(1<9 & AC=1 So, Actual LN is >F : ADD '6' to LN.)
(2<9 & CY=1 So, Actual HN is >F : ADD '6' to HN.)

1000 1001
+ 1001 1000
0010 0001 (21)H

AC=1
Cy=1

21
66
187

0010 0001

21 +66 = 187

Auxiliary Carry Flags

Eg4- 89 – 1000 1001

+89 – 1000 1001

178 0001 0010 **(12H)**

AC =1 Carry = 1

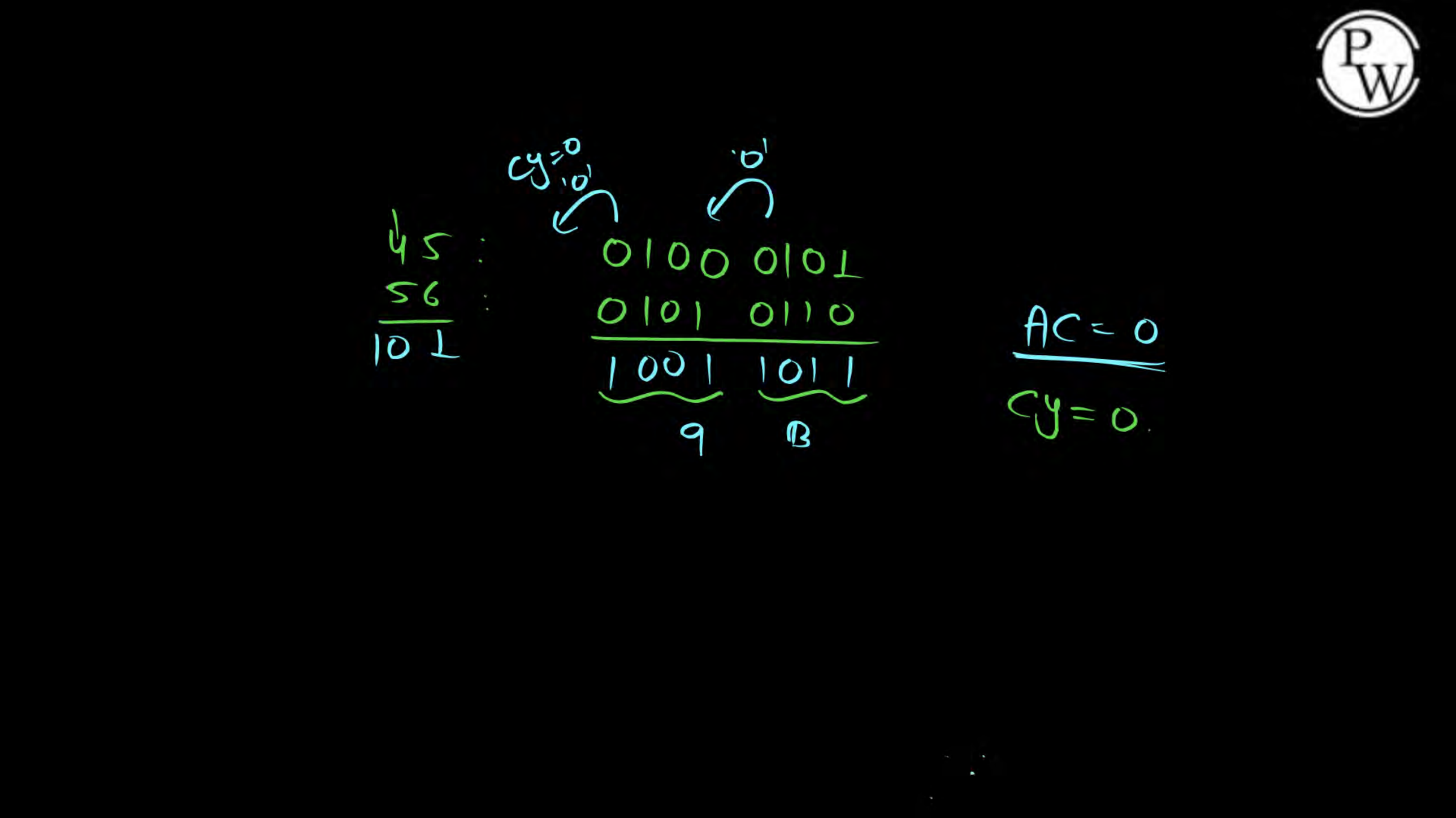
LN(18)> F, **AC=1;**

HN(17)>F, **CY=1:**

(2<9 & AC=1 So, Actual LN is >F : ADD '6' to LN.)

(1<9 & CY=1 So, Actual HN is >F : ADD '6' to HN.)

12 +66 = **178**


45 :
56 :
101
cy=0
'0'
'0'
0100 0101
0101 0110
1001 1011
9 B
AC = 0
cy = 0.

4. Zero Flags

Zero Flag: Condition is as follows:

"Is the ALU output (ACC.) value zero".

→ T = Set = 1 = Z

→ F = reset = 0 = NZ

Zero

Acc. Value	Flag status
=0	1
≠0	0

This flag is used to implement the control structure in a base hardware {if, else, while, ...} for checking the loop condition.

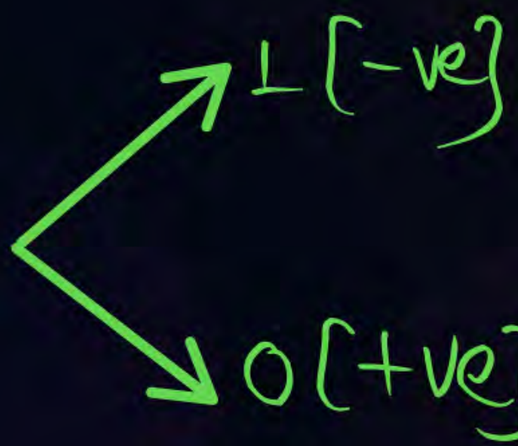
5. Sign Flags

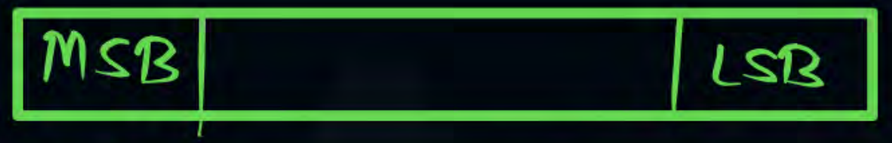
Sign Flag: Condition is as follows :

"Is the MSB bit of ALU output (ACC.) contain "1".

→ T = set = 1 = NG (– ve logic)

→ F = set = 0 = PL (+ve logic)





'1' ⇒ Sign = 1 [-ve]

'0' = Sign = 0 [+ve].

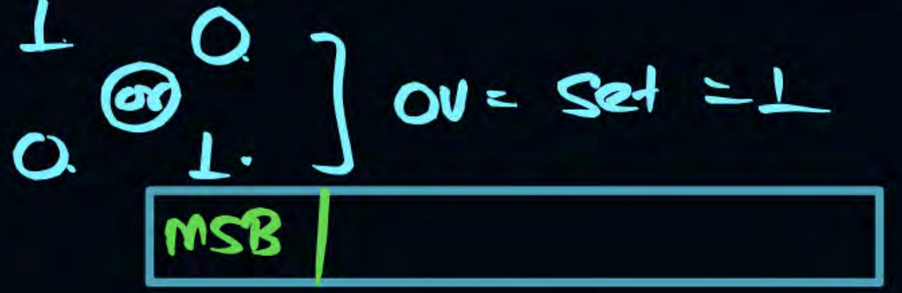
1. Carry Flag [Unsigned]
2. Parity
3. Auxiliary Carry (BCD arithmatic)
4. Zero Flag
5. Sign Flag
6. Overflow Flag.

6. Overflow Flags

Overflow Flag: Condition: is There is a carry into the MSB & No Carry Out of MSB or Vice-Versa.

→ T = set = 1 = OV

→ F = set = 0 = NOV



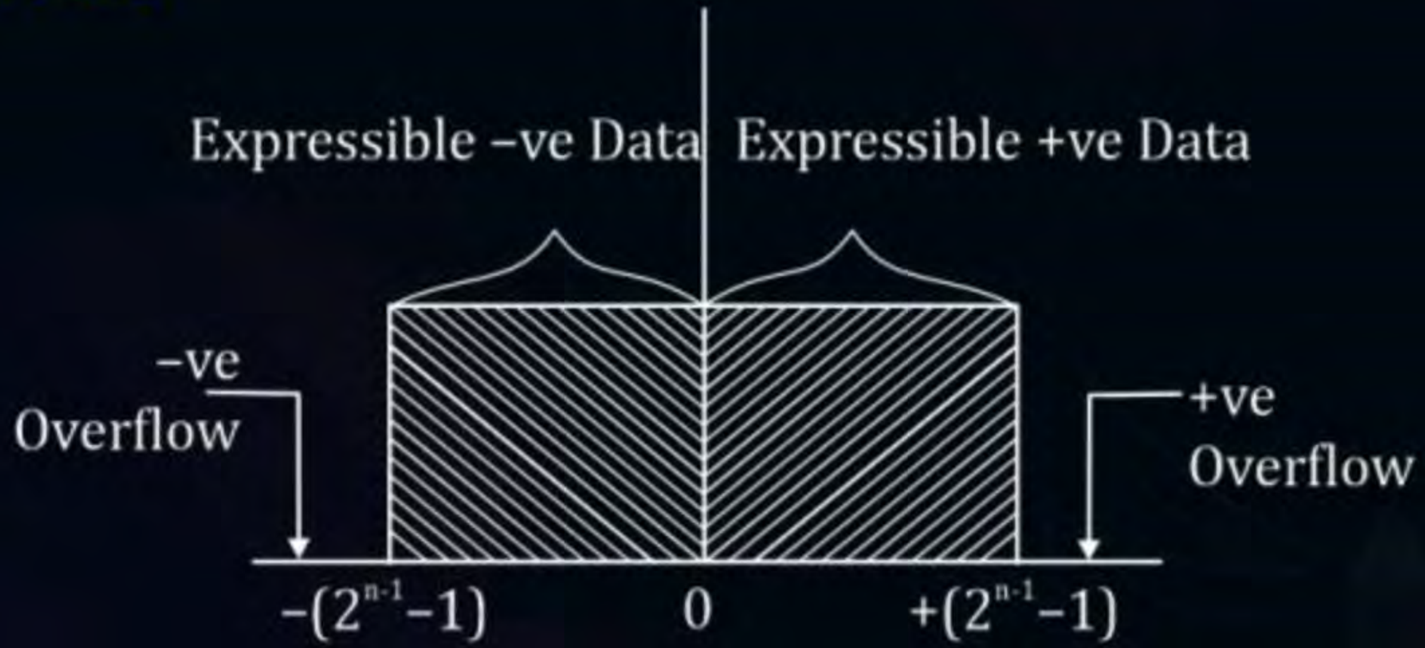
This flag is used in the **signed arithmetic** to indicate the range exceeding condition.

n bit 2's Complement Range is : $= -2^{n-1}$ to $+(2^{n-1}-1)$

4 bit Complement Range is : $\Rightarrow -2^{4-1}$ to $+(2^{4-1}-1) \Rightarrow -8$ to $+7$.

Singed DATA

- Signed data:



{4- bit : {– 8 to +7}}
{5- bit : {– 16 to +15}}

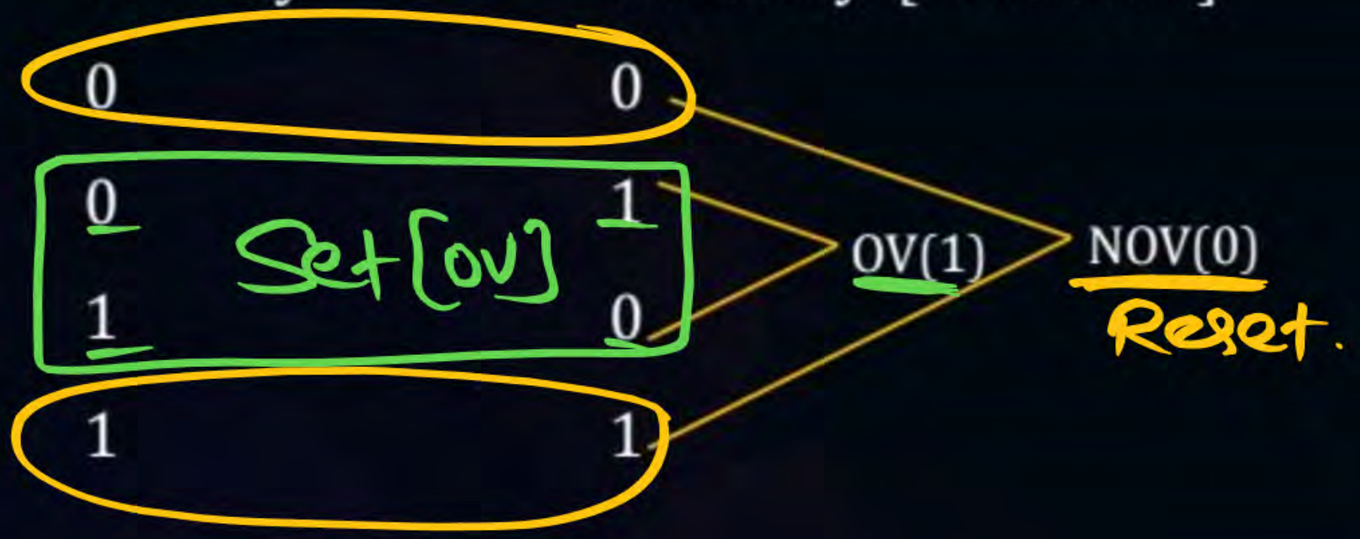
6. Overflow Flags

1. Note:- **Overflow flag is used in the CPU design to indicate the range exceeding condition of a signed**

- **Condition: "There is a carry into the MSB & No carry out of MSB or Vice versa"**
  - T = set = 1 OV
  - F = reset = 0 NOV

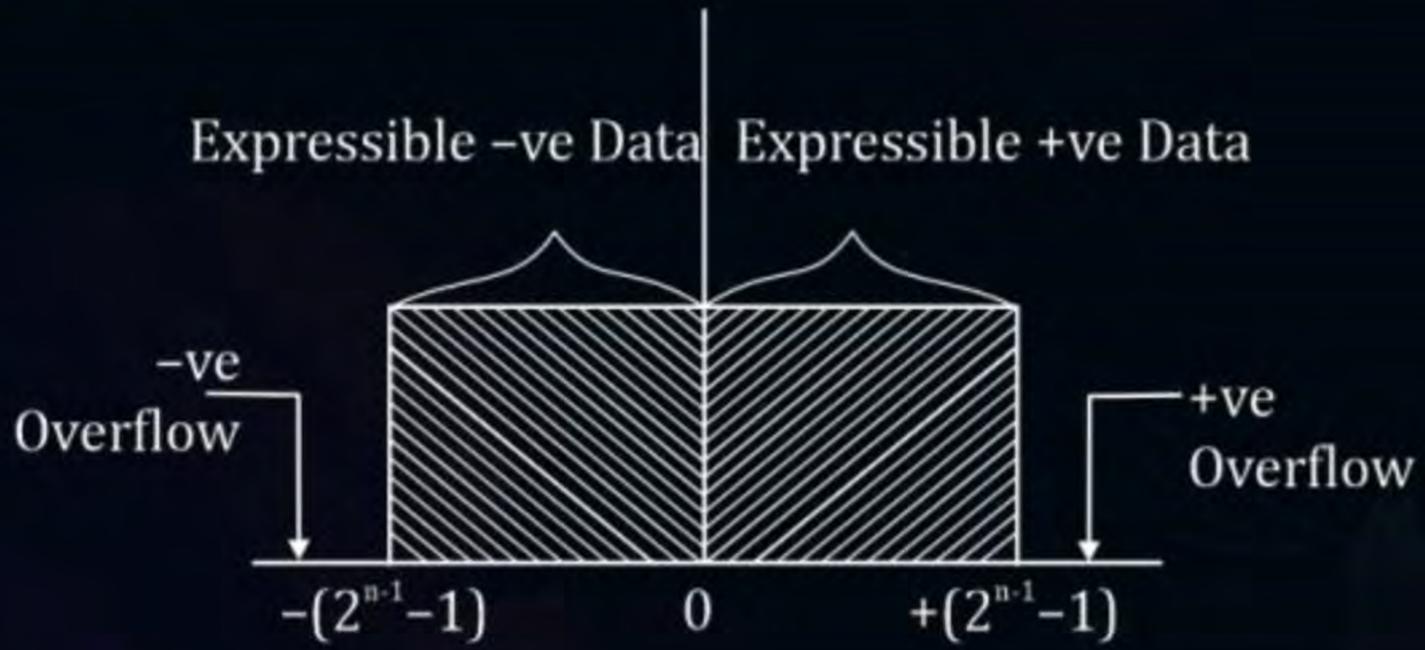
6. Overflow Flags

- Operation:- **XOR between the in-carry & out carry w.r.t. to MSB**



SIGNED DATA

- Signed data:



{4- bit : {– 8 to +7}}
{5- bit : {– 16 to +15}}

6.OVERFLOW FLAG

Condition :

$OV(x, y, z) = \underset{\Downarrow}{xyz} + \underset{\Downarrow}{\bar{x}\bar{y}\bar{z}}$

(+)positive overflow

(–)Negative overflow

Signed → +ve overflow
Signed → –ve overflow.

$OV(x,y,z) = \underbrace{\bar{x}\bar{y}z}_{+ve\ overflow} + \underbrace{xy\bar{z}}_{-ve\ overflow} = 1 \Rightarrow$ Overflow Flag is Set.

X: MSB of operand 1
Y: MSB of operand 2
Z: MSB of Result.

6. Overflow Flags

Overflow: Condition is as follows :

$OV(x, y, z) = \bar{x}\bar{y}z + xy\bar{z}$

$\bar{x}\bar{y}z$ ⇓ (+)positive overflow

$xy\bar{z}$ ⇓ (–)Negative overflow

$xyz + xyz$

xyz ⇓ (+)positive overflow

xyz ⇓ (–)Negative overflow

$OV(x, y, z) =$

- MSB of Op1
- MSB of Op2
- MSB of result

$\bar{x}\,y\,z + x\,y\,\bar{z}$

$\bar{x}\,y\,z$: + ve over flow

$x\,y\,\bar{z}$: – ve over flow

Range = −8 to +7

1	0	
0	1	set OV=1

out of MSB | MSB

+7
+7
+14

+ve overflow

0+1 ⇒ OV = set

OV=1

0 1 1 1
0 1 1 1
1 1 1 0
(14)

X=0
Y=0
Z=1

$\bar{x}\bar{y}z + xy\bar{z}$

1.1.1 + 0.0.0 = 1 (OV = set)

+ve overflow

X:1
Y:1
Z=0

−8
−8
−16

−ve overflow

1+0 ⇒ OV = set

1 0 0 0
1 0 0 0
0 0 0 0

$\bar{x}\bar{y}z + xy\bar{z}$

0.0.0 + 1.1.1 = 1

0+1 = 1

−ve overflow

SIGNED DATA

Ex-

+ 7	– 8	
+ 7	– 8	
+ 14	– 16	Overflow (–ve)

+ve Overflow

+ 7	– 6
– 4	+ 4
– 3	–2

No - overflow

+ 7	: 0 1 1 1
+ 7	: 0 1 1 1
+ 14	: 1 1 1 0
OV = 1	OY = 1

ACC - 1110 | OV L | +ve 0v Ans 1110

– 8	: 1 0 0 0
– 8	: 1 0 0 0
– 16	: 0 0 0 0
OV = 1	OV = 1

ACC - 0000 | OV 1 | –ve 0v Ans 0000 2's Complement

+ 7	: 0 1 1 1
– 4	: 1 1 0 0
+ 3	: 0 0 1 1
	0 0 1 1

$0100 \Rightarrow 1011$

$+1$

1100

$-6 \Rightarrow 0110 \Rightarrow 1's \Rightarrow 1001$

$+1$

1010 2's Complement

0+0 = Not overflow

$-6 \Rightarrow 1010$

$+4 = 0100$

1110

-2

Not overflow

③

$+7$

-4

$+3$

No overflow

$\oplus_0 = 0$ (Reset)

0111

1100

0011

Not overflow

$\bar{x}\bar{y}z + x\bar{y}\bar{z}$

$1.0.0 + 0.1.0$

$0+0 = 0$ [Reset]

Not overflow

X : 1

y : 0

z : 1

$\bar{x}\bar{y}z + xy\bar{z}$

$0.1.1 + 1.0.0$

$0+0 = 0$ Not overflow

X = 0

y = 1

z = 0

SIGNED DATA

–6	:	1 0 1 0
+4	:	0 1 0 0
–2		1 1 1 0
OV =0		OV = 0

- OV (x, y, z) = $\underline{x}\,y\,z + x\,\underline{y}\,z$

  $\underline{x}$: +ve over flow

  $\underline{y}$: –ve over flow

  x: MSB of Op1

  y: MSB of Op1

  z: MSB of result

6. Overflow Flags

Overflow: Condition is as follows :

$$OV(x, y, z) = x\bar{y}z + x\bar{y}z$$

$x\bar{y}z$ ⇓ (+)positive overflow

$x\bar{y}z$ ⇓ (–)Negative overflow

$x\bar{y}z + x\bar{y}z$

$x\bar{y}z$ ⇓ (+)positive overflow

$x\bar{y}z$ ⇓ (–)Negative overflow

$OV(x, y, z) =$

- MSB of Op1
- MSB of Op1
- MSB of result

$\underbrace{x\,y\,z}_{+\text{ ve}} + \underbrace{x\,y\,z}_{-\text{ ve}}$

+ ve over flow

– ve over flow

$+7$

$+3$

$+10$

+ve overflow

$0+1=1$

0111

0011

1010

$x: 0$

$y: 0$

$z=1$

$\bar{x}\bar{y}z + xy\bar{z}$

$1.1.1 + 0.0.0$

$1+0=1$

+ve overflow

SIGNED DATA

Ex-	+ 7	– 1 1 1
	+ 3	– 0 1 1 1
	+ 10	– 1 0 1 0
	OV =1	OV =L

Justify: PSW

Accumulator
1 0 1 0
ov + ve
0 1 0 1 0
[+10]

Flog Reg.
OV = L
{+ ve ov}

Ex-	+ 7	– 0 1 1 1
	– 3	– 1 1 0 1
	+ 4	0 1 0 0
	OV =0	OV = 0

Justify: PSW

Accumulator
0 1 0 0
0 1 0 0 [+4]

Flog Reg.
OV = 0
{+ ve ov}

SIGNED DATA

Ex-	-7	$-$ 1 0 0 1
	$+3$	$-$ 0 0 1 1
	-4	$-$ 1 0 1 0
	OV = 0	OV = 0

Justify: PSW

Accumulator	Flog Reg.
1 1 0 0	OV = 0 {+ ve ov}

Ans. 1 1 0 0 [−4]

2's complement

1 1 0 0 ® 0 0 1 1

+ 1

0 1 0 0

Ex-	-7	$-$ 1 0 0 1
	-3	$-$ 1 1 0 1
	-10	0 1 1 0
	OV = 1	OV = 1

Justify: PSW

Accumulator	Flog Reg.
0 1 1 0	OV = 1 {− ve ov}

OV

1 0 1 1 0 [−10]

1 0 1 0® 0100

+1

0 1 0 1 0

$-3 \Rightarrow 0011 \Rightarrow 1100+1 = 1101$

$0111 \Rightarrow 1000+1$

1001

1+0 = 1 OV = Set

1001

1101

0110

x:1

y:1

z=0

$\bar{x}yz + x\bar{y}\bar{z}$

0.0.0 + 1.1.1 = 1

−ve overflow

SIGNED DATA

#Q. Consider the following signed data processed on a addition circuit in the ALU. What is the status of a Sign Flag, Carry Flag, auxiliary Flag overflow Flag in the computation?

1 1 0 0 0 1 1 0
1 1 1 0 1 0 0 1

AC = 0

'0'

1 ⊕ 1

1 1 0 0 0 1 1 0
1 1 1 0 1 0 0 1
1 0 1 0 1 1 1 1

1 ⊕ 1 = 0

Nov = 0

$\bar{x}yz + x\bar{y}\bar{z}$

0.0.1 + 1.1.0

= 0

AC = 0
Cy = 1
OV = 0
S = 1

Sign Flag = 1
Carry Flag = 1
AC = 0
OV = 0

Conditional Flag

1. Carry Flag
2. Parity Flag
3. Auxilliary Carry
4. Sign Flag
5. zero Flag
6. Overflow Flag.

Control Flags

Control Flag :

Based on the status of control flags, hardware operation are controlled these are user control flag with reference to 8086 flag register, 3 control flags are present:

1. **Trap/Trap Flag :**
   - → 0 : At the time of program execution (-g); go
   - → 1 : Single step program execution (-t); trace
2. **Interrupt Flag :**
   - → 0 : Disable the INT. (DI)
   - → 1 : Enable the INT. (EI)
     {maskable interrupt only}

Control Flags

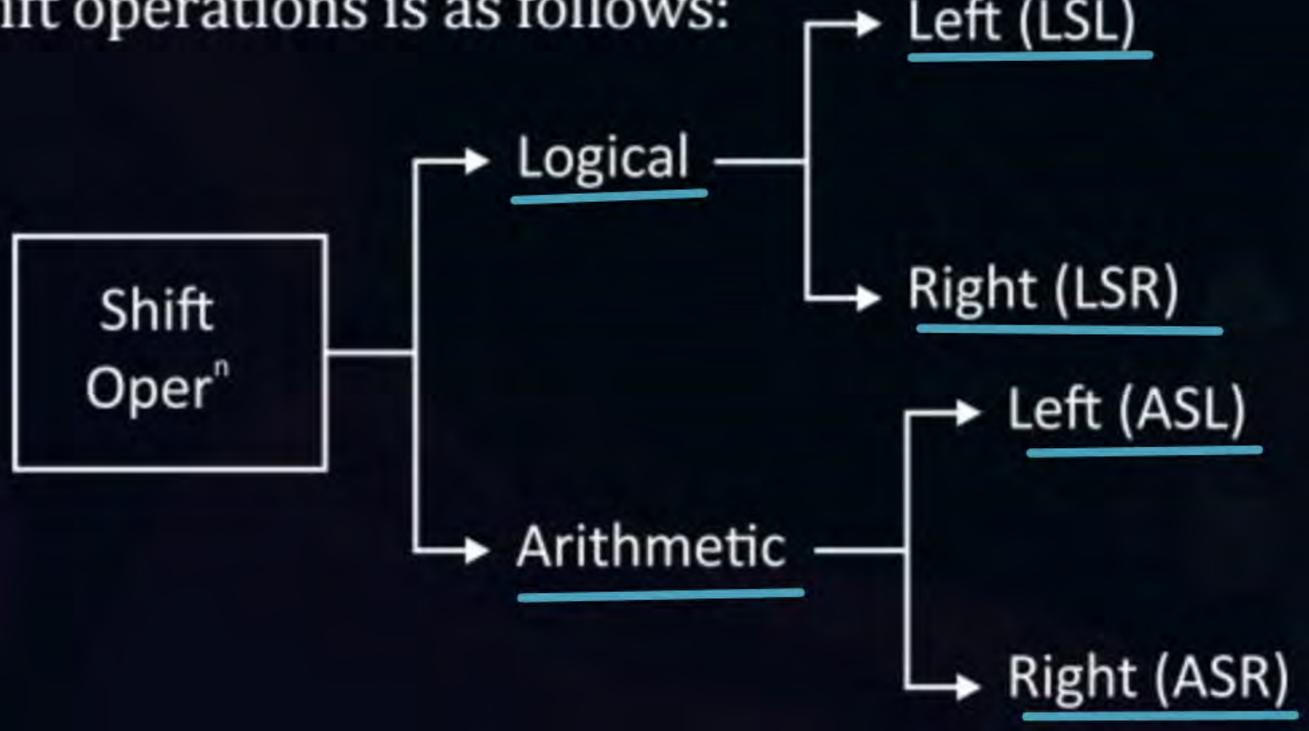
3. **Direction Flag :**
   - → 0 : Auto Increment, clear direction (CLD)
   - → 1 : Auto Decrement, set direction (STD)

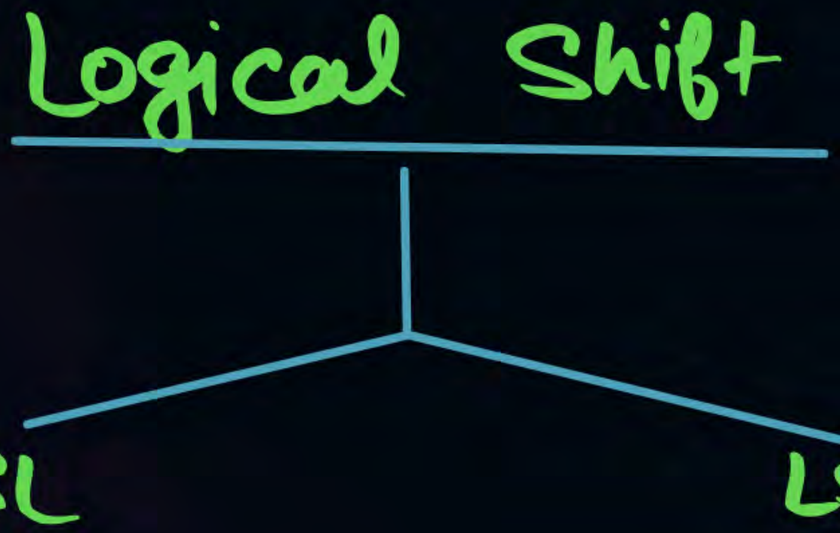
SHIFT & Rotate operation.

SHIFT OPERATION

Shift Operation:

1. While execution of this instruction, data bits are moving to right or left direction in a bitwise sequence with loss of data.
2. Different shift operations is as follows:



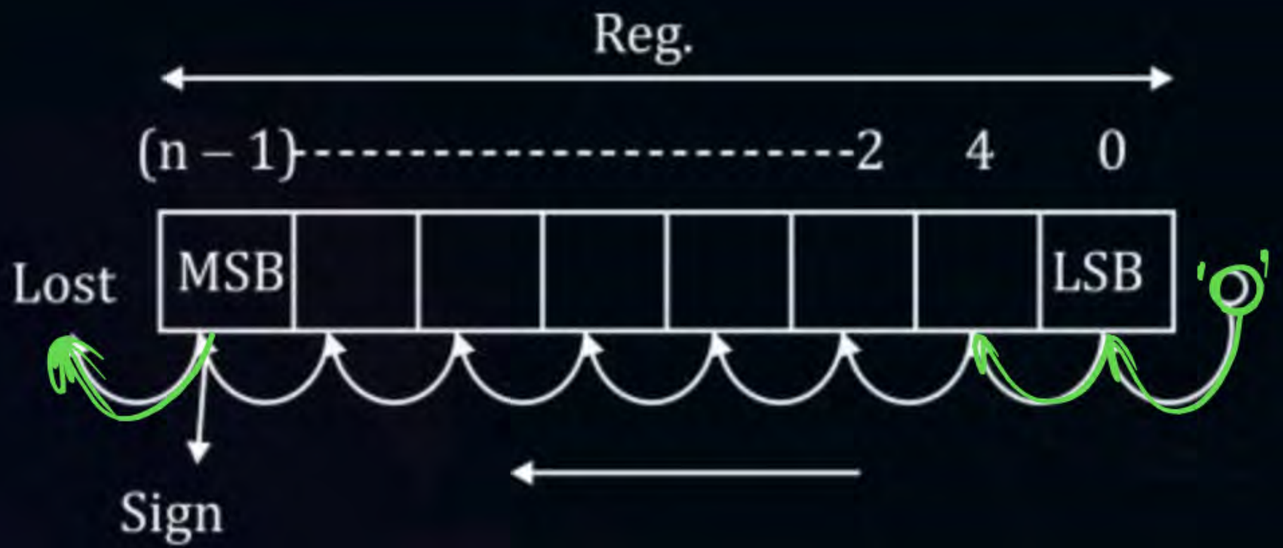

Logical Shift
LSL
LSR.

Logical Shift Operation

In a logical shift operation data bits are moving to left or right direction including the sign bit.

Logical Shift Operation

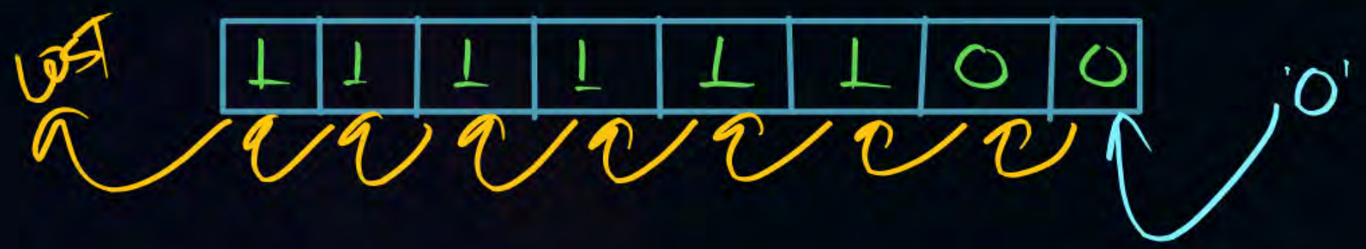
1 Logical Shift Left [LSL]



Logical Shift Operation

Logical Shift Left [LSL]

LSL r0; **r0 = (FC)H**



1 1 1 1 1 0 0 0

$(F\ 8)_{16}$

A: 10
B: 11
C: 12
D: 13
E: 14
F: 15

F: 15
1111
C: 12 ⇒ 1100

Logical Shift Operation

Logical Shift Left [LSL]

LSL r0 #4; [4 Times shift] **r0 = (FC)H**

1111 1100 '0'

I st time 1111 1000 '0'

2nd 1111 0000 '0'

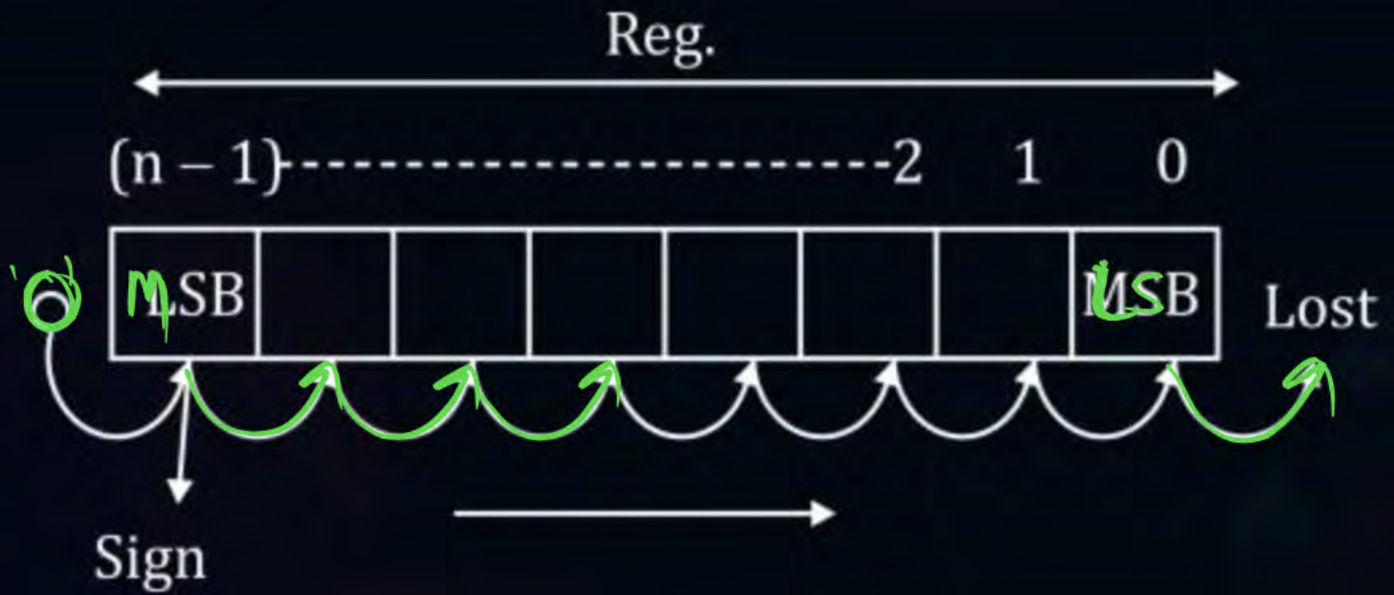
3rd 1110 0000 '0'

4th 1100 0000

(C0)

Logical Shift Operation

Logical Shift Right [LSR]



Logical Shift Operation

Logical Shift Right [LSR]

LSR r0; **r0 = (FC)H**

0 1 1 1 1 1 1 0

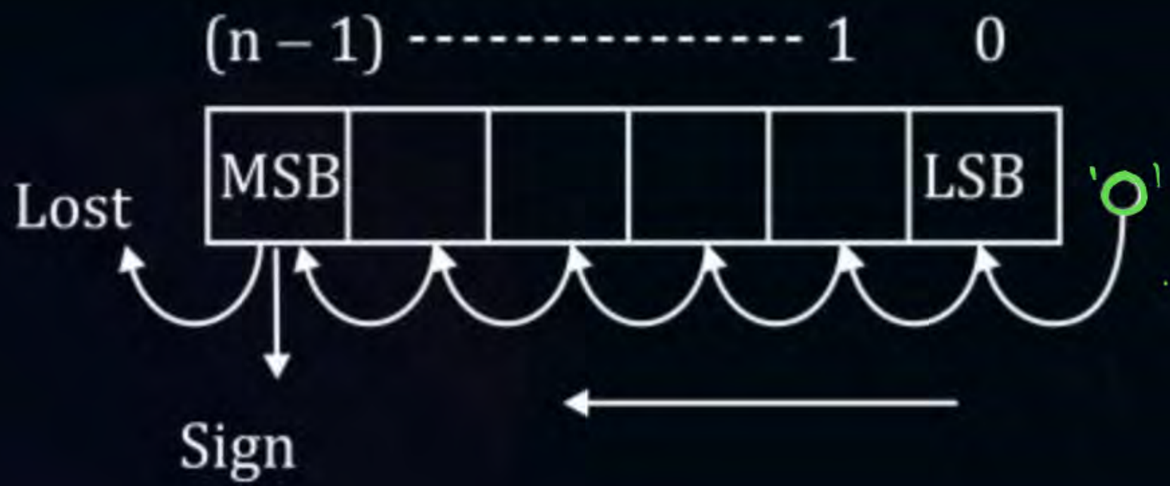
7 E

Arithmetic Shift Operation

In a arithmetic shift operation, data bits are moving to left or right direction except the sign bit.

Arithmetic Shift Operation

Arithmetic Shift Left [ASL] : Multiply a signed binary number by 2.



: Implement wise
{ASL = LSL}

Arithmetic Shift Operation

Arithmetic Shift Left [ASL]

ASL r0; **r0 = (FC)H**

Arithmetic Shift Operation

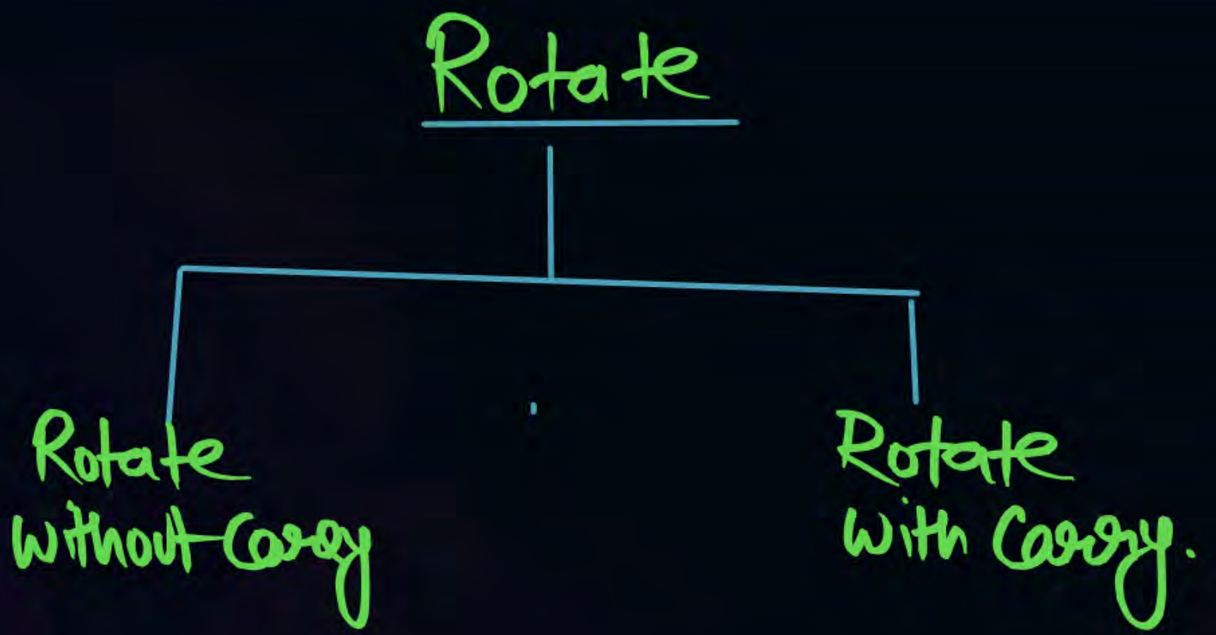
Arithmetic Shift Right[ASR] : divide a signed binary number by 2.

Arithmetic Shift Operation

Arithmetic Shift Right[ASR]

ASR r0; **r0 = (FC)H**

Rotate Operation



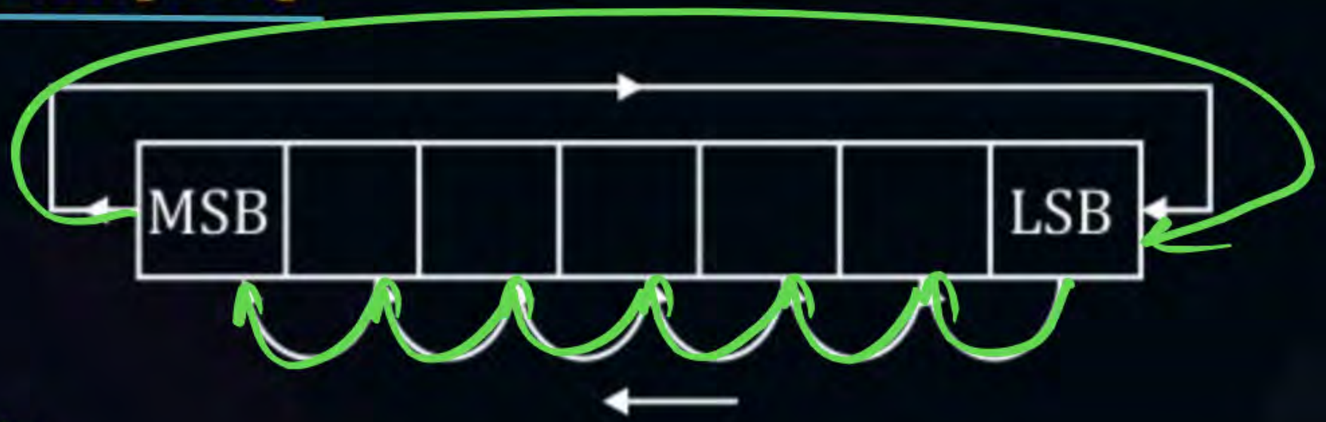
Rotate Operation

Rotate Operation : [Circular Shift Operation]

1. While execution of this instruction data bits are moving to left or right direction bit by bit without loss [Circular shift]

Rotate Operation without Carry

Rotate Operation Left [ROL]



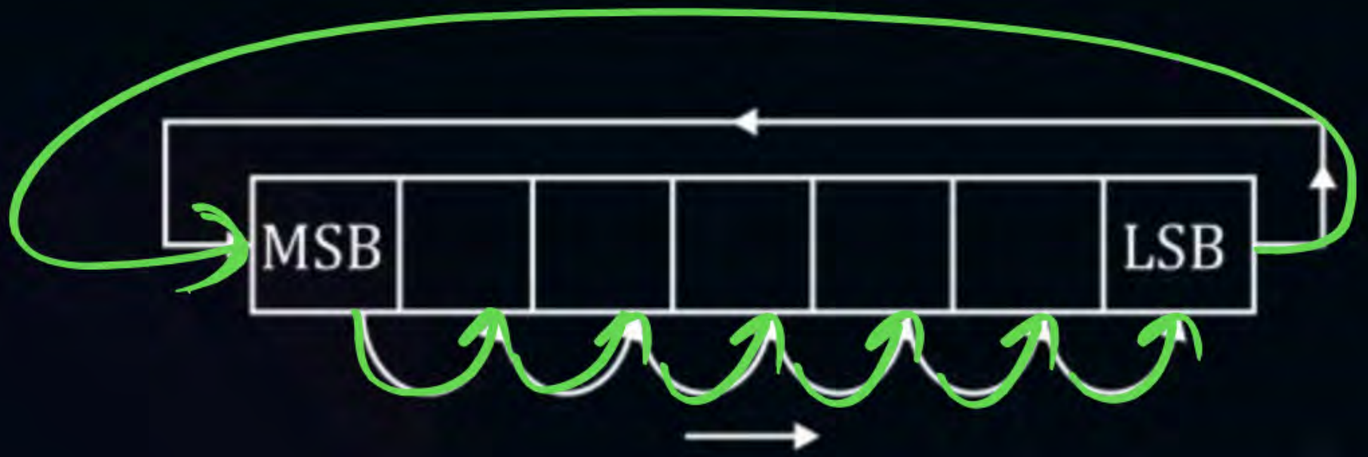
ROL %o : FCH

1111 1100

1111 1001

Rotate Operation without Carry

Rotate Operation Right [ROR]



Rotate Operation with Carry

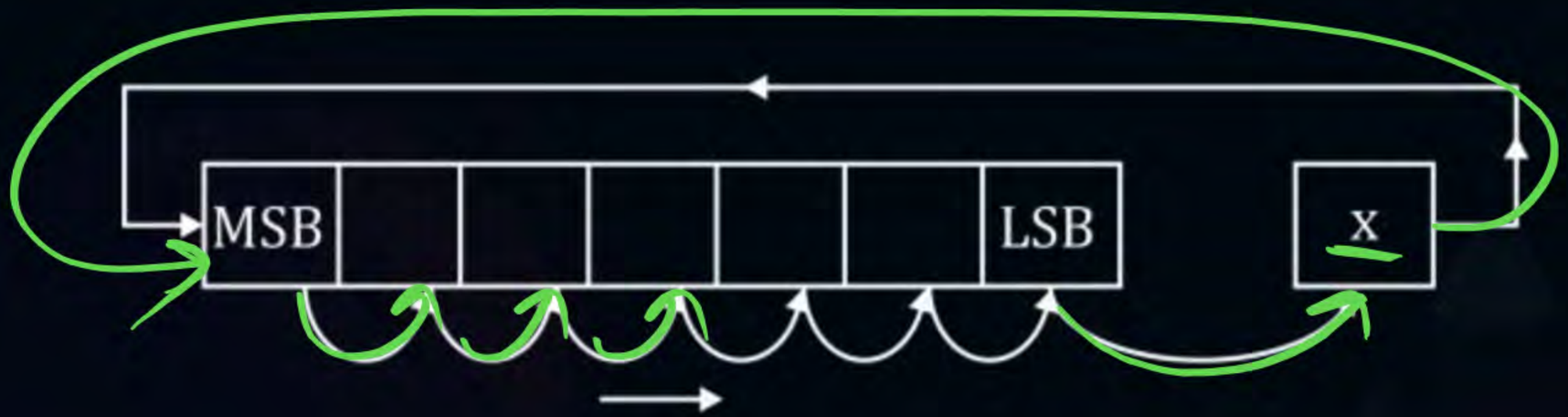
Rotate with Carry:

RRC | RCR [Rotate Right with Carry]

RLC | RCL [Rotate Left with Carry].

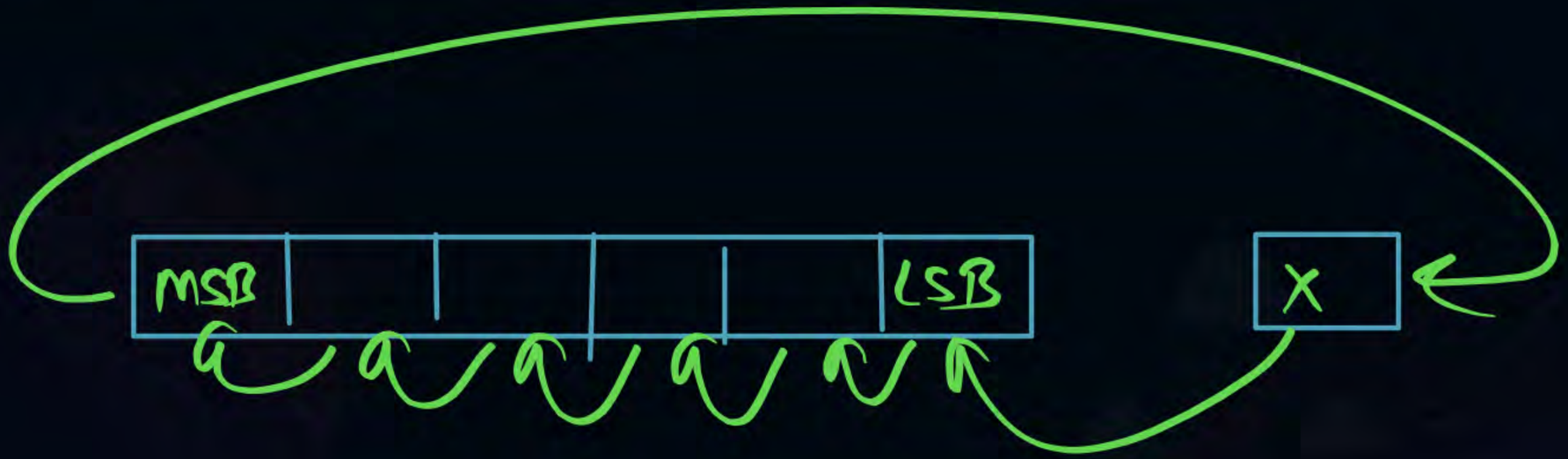
Rotate Operation with Carry

Rotate Right with Carry [RRC/RCR]



Rotate Operation with Carry

Rotate Left with Carry [RLC/RCL]

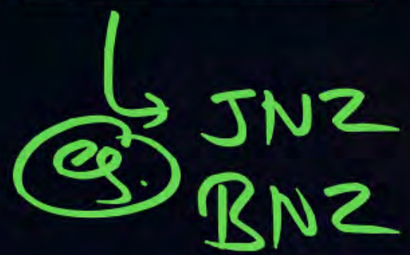


Transfer of Control

C. TOC [Transfer of Control Instruction]

i. Unconditional ⇒ Without Any Condition Control Transfer.

ii. Conditional



Transfer of Control

1. Unconditional TOC :

1. While execution of this instruction, control will be transfer to target location without checking any condition.
2. This instruction is used to implement the conditional selection statement [Go to] and machine control instruction. [Halt]

Transfer of Control

Halt :

It is a machine control instruction, invokes unconditional jump with starting address of "Halt" as a target address.

While execution of this instruction, control will be transfer to same location again and again without changing the data therefore, user perspective program execution is completed. But, CPU perspective same instruction is executed up to infinite time.

Therefore reset operation is required to execute the new program.

Transfer of Control

Conditional TOC :

1. While execution of this instruction, **associated condition is evaluated based on the status of a previous instruction.** When the condition true then control will be transfer to target location otherwise sequential instruction in the program is executed.
2. This instruction is used to implement the conditional selection statement [if switch] and various iterative statement [for, while do-while]

[CALL – RET]
[Push – POP]

Transfer of Control

Subprogram : Definition:

Subprogram is a reusable program means frequently occurred functionality is developed as a subprogram only once later refer it in the main application mainly times.

Characteristics :

1. Single entry point – single exit point
2. Main program is suspended during the execution of a sub program
3. Control will be transfer back to main program of a subprogram.

IMPLEMENTATION

Implementation :

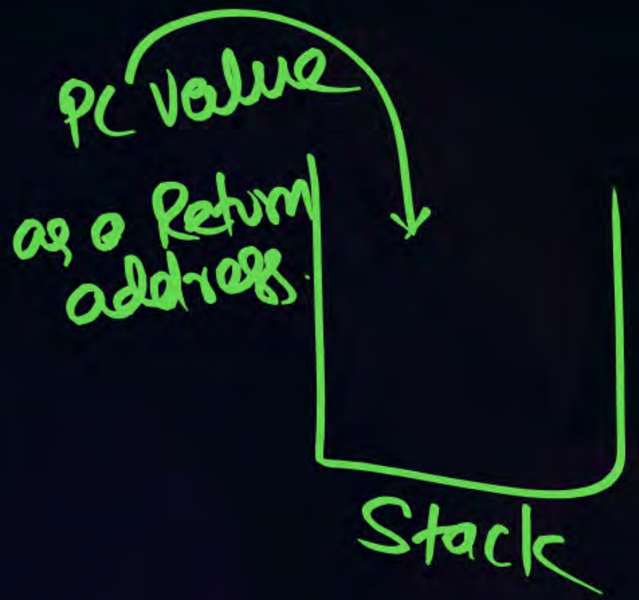
CALL – RET

CALL → Entry → Present in main prog. → Invokes push oper^{n} to save PC value

RET → Exit → Present in sub prog. → Invoke pop oper^{n} to restore the return addr. to PC.

IMPLEMENTATION

a. In this implementation process stack is used to store the return address.
b. Return address is a next instruction address after the CALL instruction.



INTERRUPT CYCLE

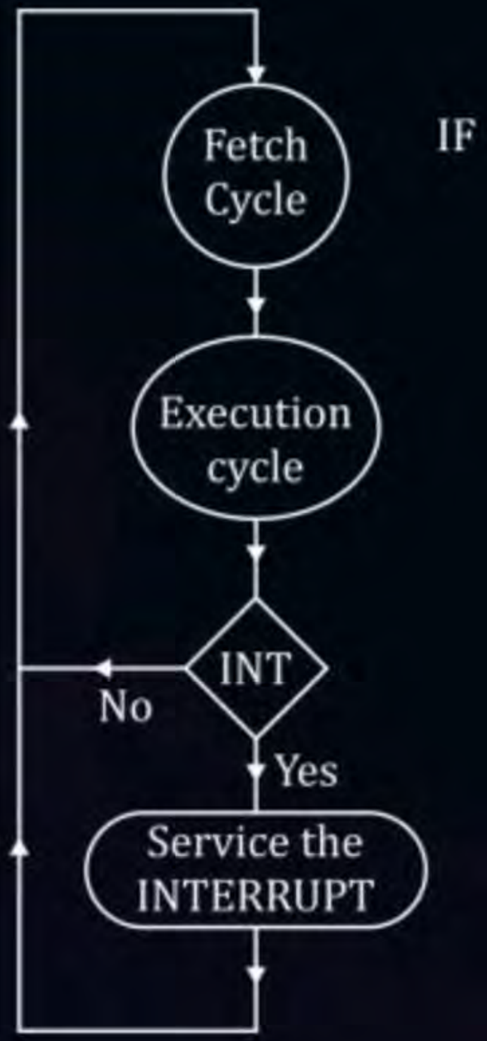
Interrupt Cycle :

1. Interrupt is a signal which is generated by component in the computer [I/O]
2. Interrupt cycle is connect at the end of a execution cycle so, CPU will be respond to a interrupt only after the completion of a current ins^{n} execution.

INTERRUPT CYCLE

Interrupt Cycle :

① Fetch cycle
② Execute cycle
③ Interrupt cycle.



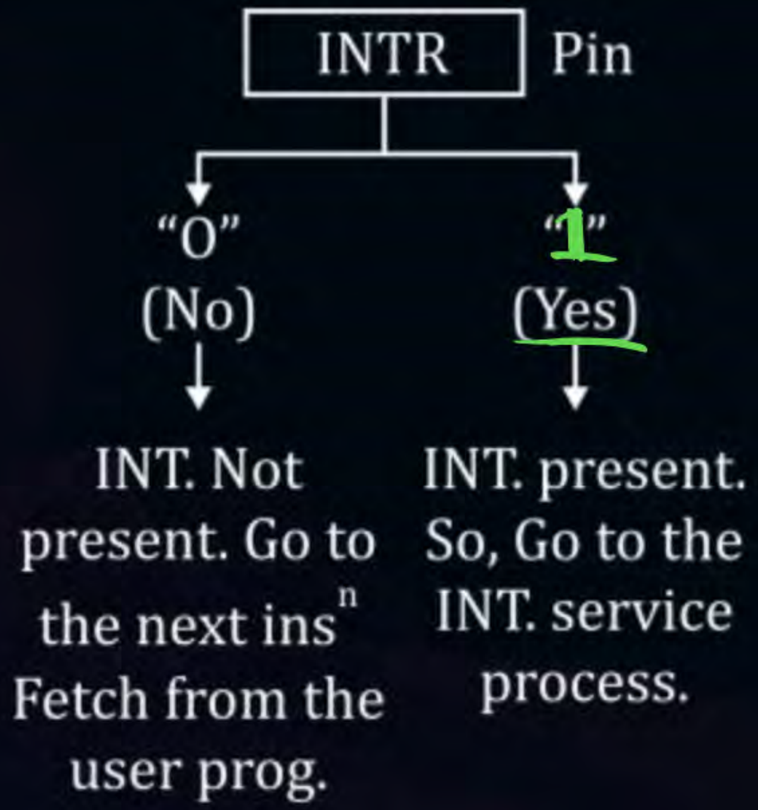
IF

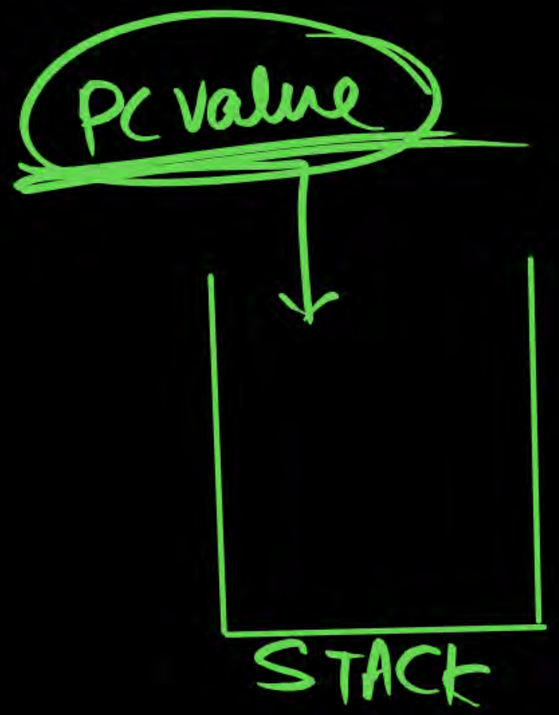
ID
OF
PD
WB
} Decode Operand Fetch Process Data Write Back.

Interrupt cycle

INTERRUPT CYCLE

3. CPU reads the status of a interrupt pins to detect the interrupt after completion of a current instruction execution.




PC value
STACK

INTERRUPT CYCLE

4. When the CPU detect the interrupt then it save the "PC" value into a stack later transfer the control to "interrupt vector table". (IVT)
5. IVT is a part of memory which contain interrupt subprogram address.
6. Interrupt subprogram referred with a "vector address", end with a IRET RETI/REF instruction.
7. IRET instruction invokes the pop operation to restore the return address into a "PC". Therefore, after service the interrupt control will be transfer to a user program.

NOTE :

Objective of a interrupt cycle is interrupt subprogram initializing. In this process, "PC" saved into a stack, later vector address, loading into a 'PC'.

INTERRUPT CYCLE

Two approach are present to service the interrupt

1. Single interrupt processing [Non-pre-emption]
2. Nested interrupt processing [Pre-emption]

INTERRUPT CYCLE

ISRO

Q. A CPU has 4-interrupts [I_1, I_2, I_3, I_4] with a respective service times of 40 ns, 30 ns, 20 ns, 10 ns. Response time of a interrupt is 5 ns. I_1 has the highest priority and I_4 has the least priority. What is the range of time required to service I_4 interrupt when the interrupt may or may not occur simultaneously ?

Response + Service

Only I_4: 10 + 5 = 15 ns.

I_1, I_2, I_3, I_4: [(40+5), +(30+5)+(20+5)+(10+5)]

45 + 35 + 25 + 15

120

15 ns to 120 ns

(15 to 120) ns

Type of Interrupt.

INTERRUPT CYCLE

Types of Interrupt :

1. Hardware interrupt :

i. It is a signal which is generated by the hardware component. Hardware component are interfaced to CPU via internal interface and external interface so, hardware interrupts are internal interrupt and external interrupt.

ii. External interrupt is a signal which is generated by the external hardware.
Eg. : Power supply, Basic I/O devices [Keyboards, Printer, mouse, etc]

iii. Internal interrupt is a signal which is generated by the internal hardware.
Eg. : Temperature sensor, Timer clock, Critical Sensor in the motherboard, Invalid opcode, Divide-by-zero, Stack overflow etc.

iv. In the CPU design hardware pins are reserved to hold the hardware interrupts.

INTERRUPT CYCLE

iv. In the CPU design hardware pins are reserved to hold the hardware interrupts.

Eg. In 8085 Microprocessor

(TRAP) RST 4.5 : HIGH Priority
RST 7.5
RST 6.5
RST 5.5
INTR : LOW Priority

Eg. In 8086 Microprocessor

NMI : HIGH Priority
INTR : LOW Priority

NMI : Non Maskable Interrupt.

INTERRUPT CYCLE

2. **Software Interrupts :**

It is an instruction, used to execute the predefine IO function in the processor mode.

Ex. : System Calls.

3. **Maskable Interrupts :**

It is a low priority interrupt so, CPU may or may not service the interrupt. Therefore all the low priority IO devices are connected to a CPU via maskable interrupts pins.

4. **Non-Maskable Interrupts :**

It is a high priority interrupts so, CPU compulsory service the interrupts. Therefore high priority hardware is connected to the pins.

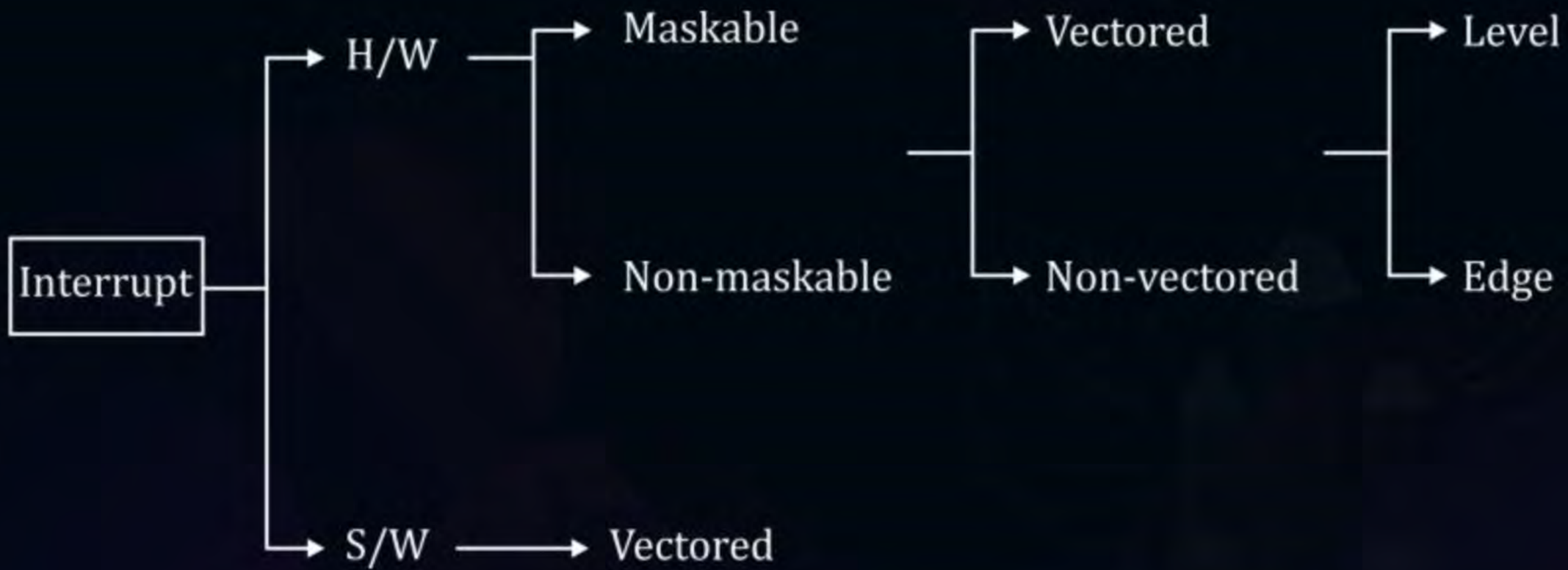
5. **Vectored Interrupts :**

This interrupts contain vector information so vector address is calculated based on the interrupt vector.

INTERRUPT CYCLE

6. **Non-Vectored Interrupt :**
   This interrupt does not contain the vectors so, CPU enable the acknowledgment and waiting until the interrupt source supplies the vectored address.
7. **Level Triggered Interrupts :**
   This interrupt enables based on the level of a clock signal [High level, low level]
8. **Edge Triggered Interrupts :**
   This interrupts enables based on the raising edge transition or falling edge transition of clock signal.

INTERRUPT CYCLE



1. Flag
2. Shift & Rotate operation
3. Interrupt cycle.
4. Amdhals Law & Flynn's classification.

PERFORMANCE ANALYSIS

1. Performance is an indirect measurement depends on the execution time.
2. Time required to complete the program is called as execution time.
3. Execution time means CPU time i.e.,

CPU time = # seconds/programming

↓

Instruction/programming

↓

Instruction Count (IC)

IC x CPI x Cycle

- # Cycle/Instruction → CPI
- # Seconds/Cycle → Cycle Time

PERFORMANCE ANALYSIS

4. In the program different instruction take different cycles to complete therefore, program execution time is calculated as-

CPU time = Σ (IC_i * CP_i) * Cycle time

i : Types of Instruction.

PERFORMANCE ANALYSIS

5. Speed-up (s) factor is used to measure the performance gain i.e.,

$$S = \frac{\text{Performance X}}{\text{Performance y}}$$

$$S = \frac{\frac{1}{ET_X}}{\frac{1}{ET_Y}}$$

$$\boxed{S = \frac{ET_Y}{ET_X}}$$

S = 'n' → some constant

{System 'X' will run "n" times faster than system 'Y'}

$$\text{Performance} \propto \frac{1}{ET}$$

AMDHAL'S LAW

1. In the processor design, different quantitative principles are present, used to improve the performance.
2. One of the principle state that improve the performance within a cost and price limit by making the common case fast called as **Amdhal's Law**.
3. **Amdhal' s Law** concentrate on the performance gain after enhance the part of a system i.e.,

$$S_{overall} = \frac{\text{Performance of a system with enhance (New)}}{\text{Performance of a system without enhance (Old)}}$$

$$S_{overall} = \frac{1/ET_{New}}{1/ET_{Old}}$$

$$S_{overall} = \frac{ET_{Old}}{ET_{New}} \quad \ldots\ldots\ldots(1)$$

AMDHAL'S LAW

4. In the enhancement process, only the part of the system is modified so, new system contain two portion's ____.
   (i) Unenhanced Portion
   (ii) Enhanced Portion
   ET_{new} = ["ET" of unenhanced portion] + ["ET" of enhanced portion]
5. To calculate ET_{New}, two factors required-
   (i) Fraction enhance (F)
   (ii) Speed up enhance (S)
6. Fraction enhance (F) indicates, how much portion of the system is modified i.e.,
   F: Enhanced portion
   (1 – F): Unenhanced portion

AMDHAL'S LAW

7. Speedup enhance(S) indicates, performance gain of an enhanced portion-

$$S = \frac{\text{Performance of new "F"}}{\text{Performance of old "F"}}$$

$$S = \frac{\frac{1}{\text{ET of new "F"}}}{\frac{1}{\text{ET of old "F"}}}$$

$$S = \frac{\text{ET of old "F"}}{\text{ET of new "F"}}$$

$$\text{ET of new “F”} = \frac{\text{ET of old "F"}}{S}$$

AMDHAL'S LAW

8. Substitute the above value in equation (2)

$$ET_{new} = ET \text{ of old } (1- F) + \frac{ET \text{ of old "F"}}{S}$$

9. Substitute the above value in equation (1)

$$S_{overall} = \frac{ET_{old}}{ET \text{ of old } (1-F) + \frac{ET \text{ of old "F"}}{S}}$$

$$S = \left[(1-F) + \frac{F}{S}\right]^{-1}$$

10. To cover the complete system, let us consider relative data i.e., system = 100%

$$S_{overall} = \frac{100\%}{(100\% - F) + F/S}$$

$$S_{overall} = \frac{1}{(1-F) + F/S}$$

$$S_{overall} = \left[(1 - F) + \frac{F}{S}\right]^{-1}$$

F: Enhanced Portion

S: Speed up Factor.

When $ET_{old} = 1$ then

$$ET_{new} < 1$$

$$\therefore \quad S_{overall} > 1$$

AMDHAL'S LAW

11. When the system contains multiple enhancement then performance gain is calculated as-

$$S_{overall} = \left[(1 - \sum F_i) + \sum \frac{F_i}{S_i}\right]^{-1}$$

i : #enhancement

F: Enhanced Portion

S: Speed up Factor

HIGH PERFORMANCE CPU DESIGN

1. High performance CPU exhibits the concurrency.
2. Concurrency means two or more instruction execution at a time.
3. According to **Flynn's classification**, four computer architecture are present.
    1. **SISD [Single Instruction Stream and Single Data Stream]**
    2. **SIMD [Single Instruction Stream and Multiple Data Stream]**
    3. **MISD [Multiple Instruction Stream and Single Data Stream]**
    4. **MIMD [Multiple Instruction Stream & Multiple Data Stream]**

HIGH PERFORMANCE CPU DESIGN

CU: Control Unit

PU: Processing Unit (ALU)

MU: Memory Unit

IS: Instruction Stream

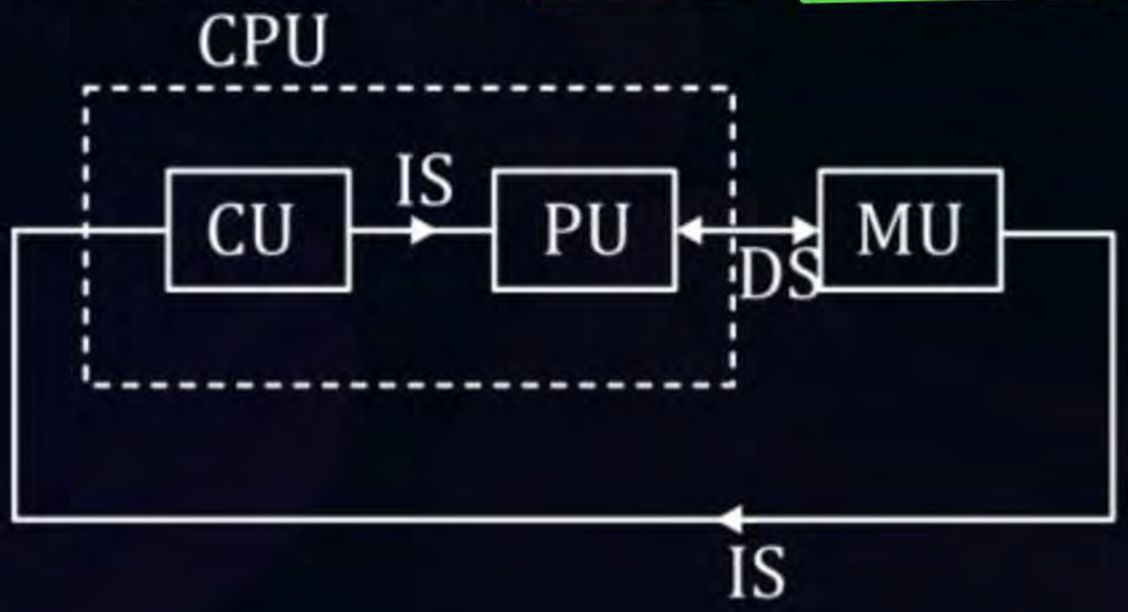
DS: Data Stream

SISD

In this architecture single processor executes a single instruction stream to operate on data stored in a single memory. Uniprocessors fall into this category.

or

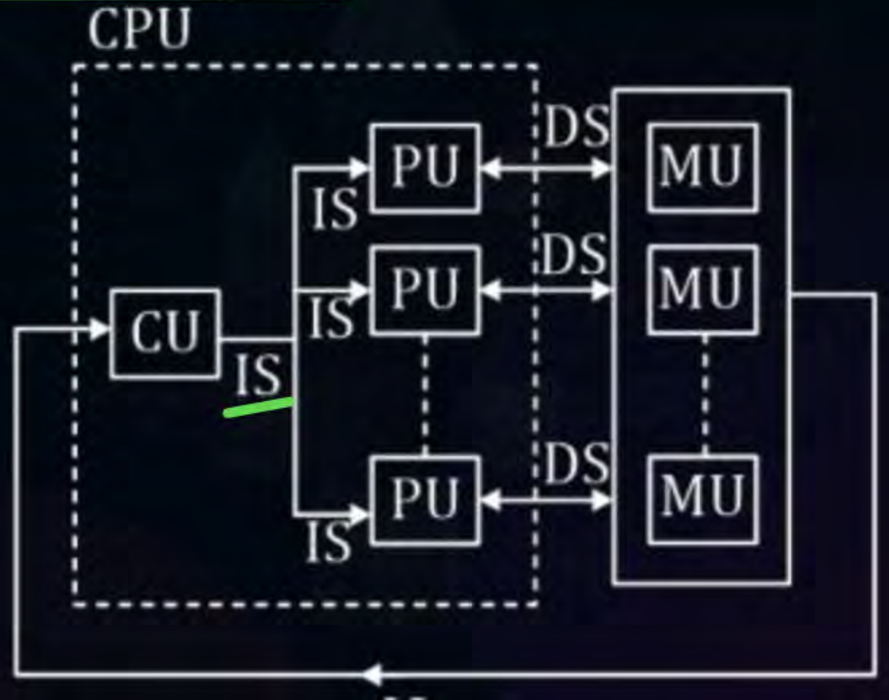
Instruction & Data both are present in Same memory & result also stored in same memory. Eg: 8085,8086, IBM 704, VAX 11/780,CRAY1.



SIMD

A single machine instruction controls the simultaneous execution of a number of processing elements on a lockstep basis. Each processing element has an associated data memory, so that instruction are executed on different sets of data by different processors. **Vector and Array processors fall into this category.**

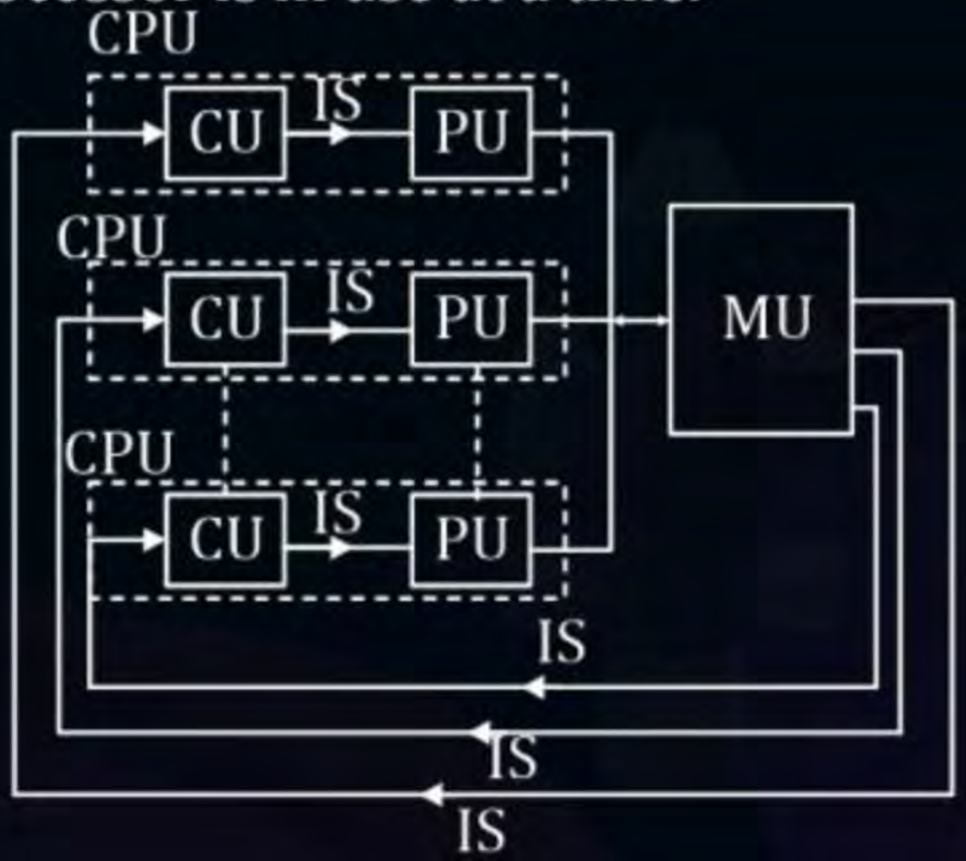
Eg. STARN Processor, MPP etc.



MISD

A sequence of data is transmitted to a set of processors , each of which execute a different instruction sequence . This structure is not commercially implemented.

It contains multiprocessor but only one processor is in use at a time.
This architecture is not yet implemented.

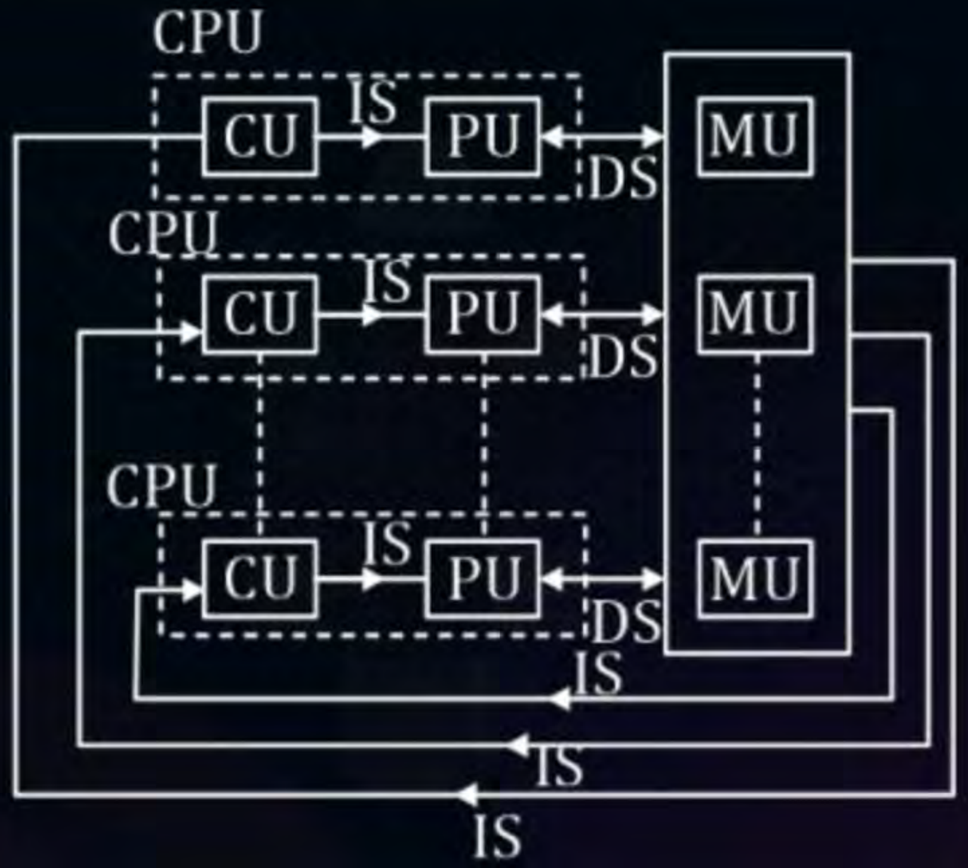


MIMD

A set of processors simultaneously execute different instruction sequences on different data sets. SMPs, clusters , and NUMA systems fit into this category.

Implemented as a multiprocessor system design.

Eg. CRAY Processor, IBM 370/168M.



System Bus

- System Bus:-

(1) Bus is a communication media/channel.

(2) Limitation of a bus is only one transmission at a time.

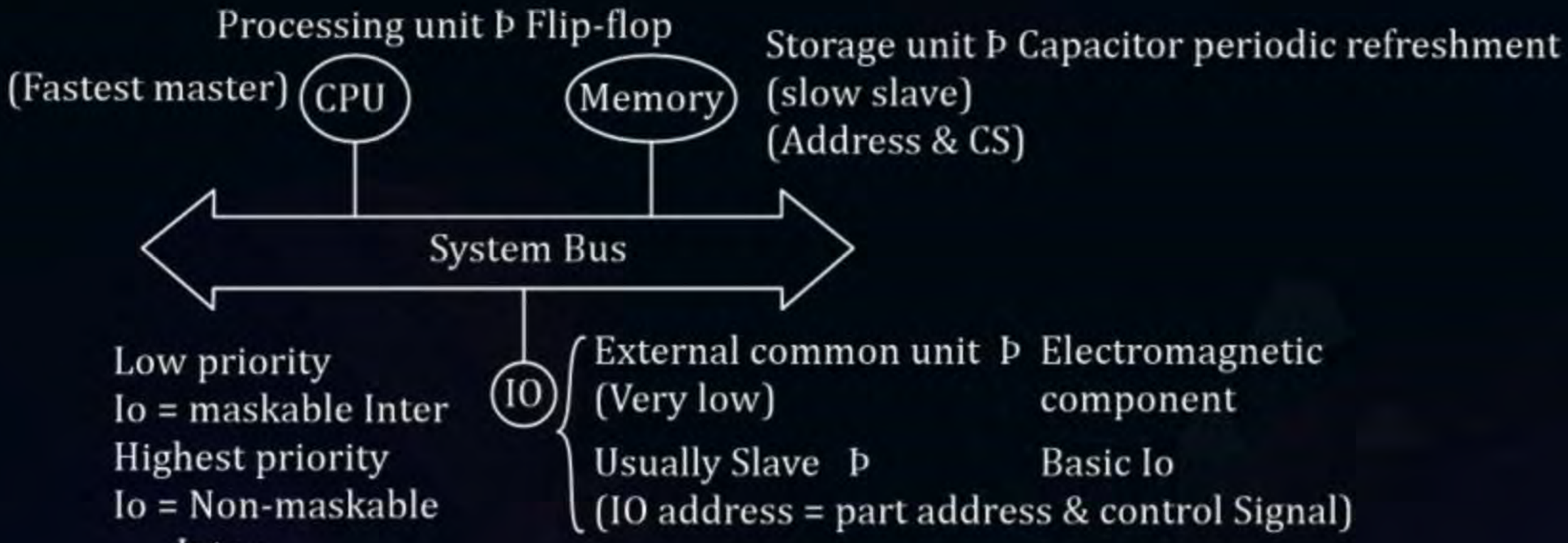
(3) A bus which is used to provide the communication between the major components of a computer is called as "system Bus".

Address line

Data line

Control line

System Bus


Processing unit Þ Flip-flop
(Fastest master)
CPU
Memory
Storage unit Þ Capacitor periodic refreshment
(slow slave)
(Address & CS)
System Bus
Low priority
Io = maskable Inter
Highest priority
Io = Non-maskable
Inter
IO
External common unit Þ Electromagnetic
(Very low) component
Usually Slave Þ Basic Io
(IO address = part address & control Signal)

Address Line

5. System bus contain three ceterones of the line used to provide the communication name as-
   (a) Address lines (Bus)
   (b) Data lines (Bus)
   (c) Control lines (Bus)

- **Address lines:-**
  (1) These lines are used to carry the address to memory & IO So, address lines are unidirectional.
  (2) Based an the width of a address bus, we can measure the capacity of a memory system.

DATA LINES

- Data lines:-
  1. These lines are used to carry the used to carry the binary sequence between the CPU, memory and IO So, Data lines are BI– Directional.
  2. Based on the width of a Data Bus, we can measure the word length of a CPU. Based on the word length, we can measure the performance of a CPU.

Example: In 8085 μp

$AD_7 - AD_0$

↓

8 DL's

↓

word length = 8 bit

↓

Operation are performed on a 8-bit data format

DATA LINES

Example:- In 8085 μp

AD_{15} – AD_0

↓

16 DL's

↓

word length = 16

↓

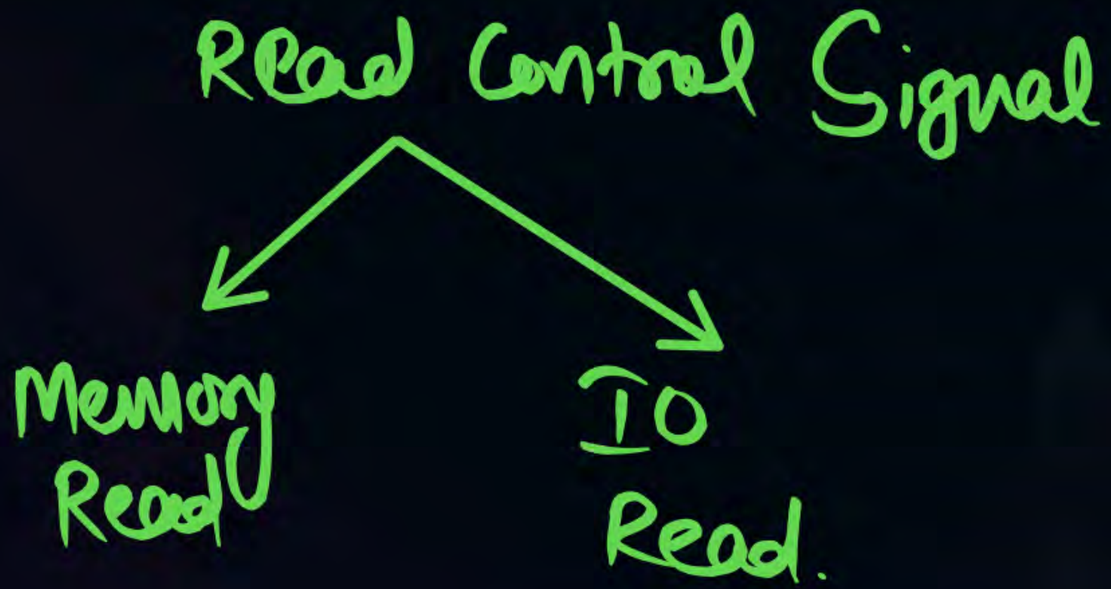
operations are performed on a 16- bit data format.

CONTROL LINES

Control Lines:-

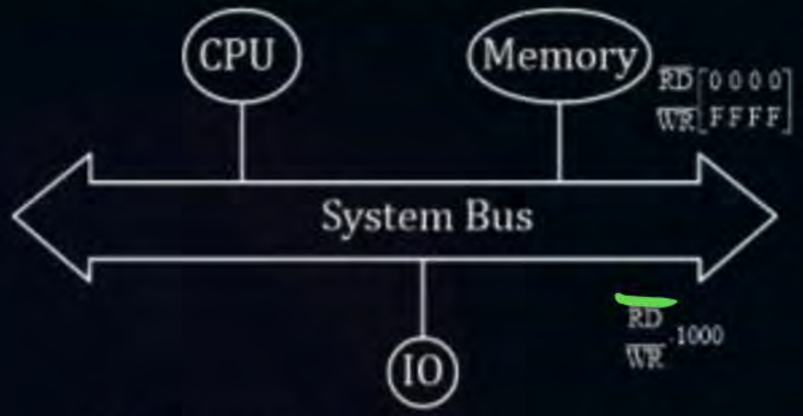
(1) These lines are used to carry the control signal and timing signals.

(2) Control signal indicates the types of operation, and timing signal used to synchronize the memory & IO operation with a CPU clock.


Read Control Signal
Memory Read
IO Read.

CONTROL LINES

NOTE:- When there is a common bus, common control signal & common address space is used between the memory & IO then there is a possibility of ambiguity (conflict).



CPU generates the memory request

1000	$\overline{RD}$

ß

1000 ® AL's

$\overline{RD}$ ® CL's

- Memory controller — 1000 – valid $\overline{PO}$ – valid
- IO controller — 1000 – valid, RD – valid

CONTROL LINES

- To handle the above problem bus configuration are used in the computer design.
  These are three kinds-
  1. IOP (Input –output process)
  2. Isolated IO (IO-mapped - IO)
  3. memory – mapped – IO
- 1. **IOP**: This configuration uses the common control signal & common Address but different buses for both memory & IO.
  Additional hardware is required bus & IO bus, it is expensive. Therefore suitable for "super computers".

Note

IOP: Common Control signal & Address. But Different Buses for Memory & IO.

CONTROL LINES

- 2. **Isolated IO (IO-mapped - IO)**

  Isolated IO:- (1): This configuration uses the common bus & common address space but different control signal for both memory & IO.

  It is less expensive ,used in general purpose computers.

Common Bus & Address Space But Different Control Signal for Memory & I/O.

CONTROL LINES

- Control Signal
  - Memory based
    - $\overline{MEMRD}$: Load
    - $\overline{MWMWR}$: Store
  - IO based
    - $\overline{IORD}$: In
    - $\overline{IOWR}$: out

Active low Pin

$\overline{\text{Pin Name}} \Rightarrow$ i/p $\Rightarrow \overline{0}$ then Enabled.

CONTROL LINES

IO/M: Hardware Pin is used to implement the Isolated IO.

IO/M	$\overline{RD}$	$\overline{WR}$	CS
0	0	1	$\overline{MEMRD}$
0	1	0	$\overline{MEMWR}$
1	0	1	$\overline{IORD}$
1	1	0	$\overline{IOWR}$

Memory

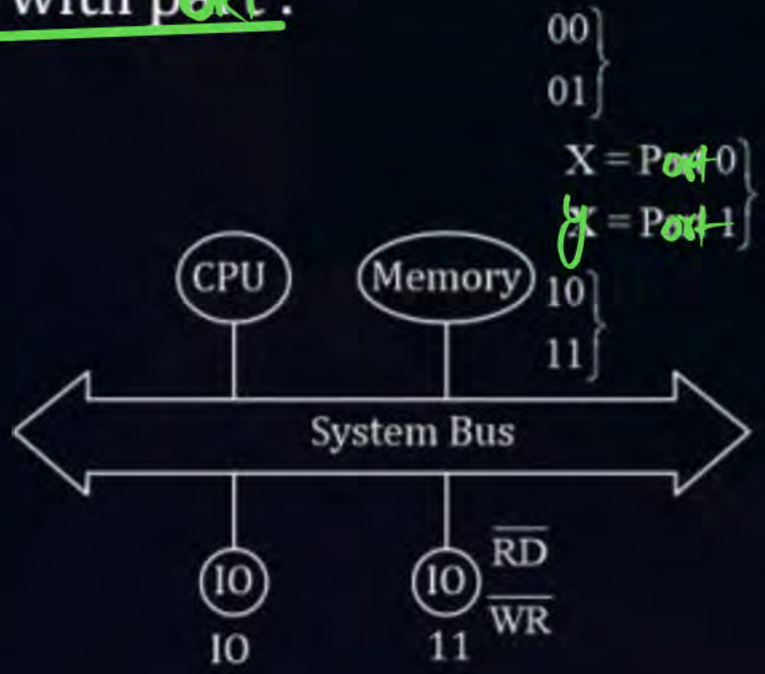
IO

MEMORY- MAPPED IO

3. Memory- mapped IO:-

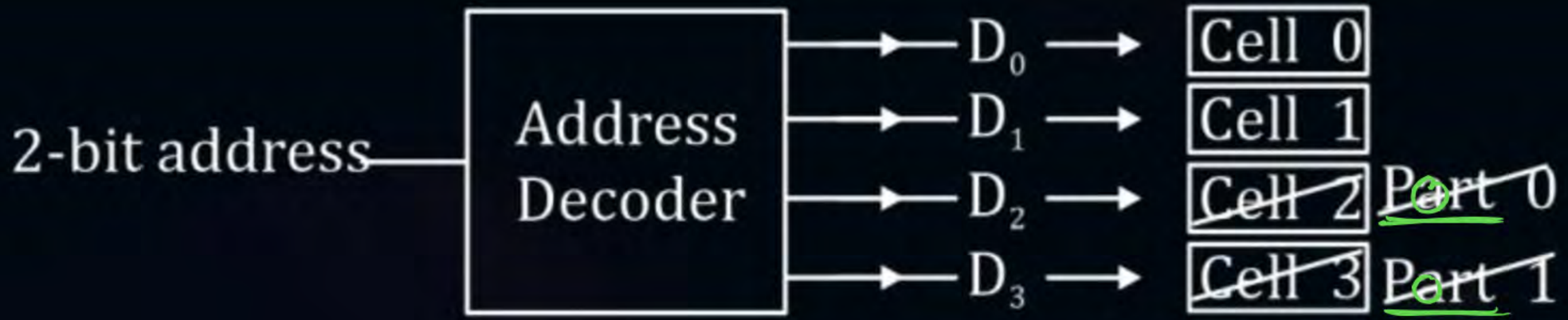
(1) This configuration uses the common bus and common control signal but unique address space for both memory & IO.

(2) In this design memory address space is shared to IO ports so, limitations "Complete memory space is not in the use because of some of the cell are replaced with port".



Common Bus & Common Control Signal But Unique Address Space for Both memory & IIo.

MEMORY- MAPPED IO



MEMORY- MAPPED IO

#Q. Consider a 32 bit hypothetical processor which support 256KB memory space. Computer is design with a memory mapped-IO in which when the 3-MSB bit of MSB is **'1'** then assign then to a **IO ports**, & remaining are using for Memory. How many port address and memory address are possible in the computers.?

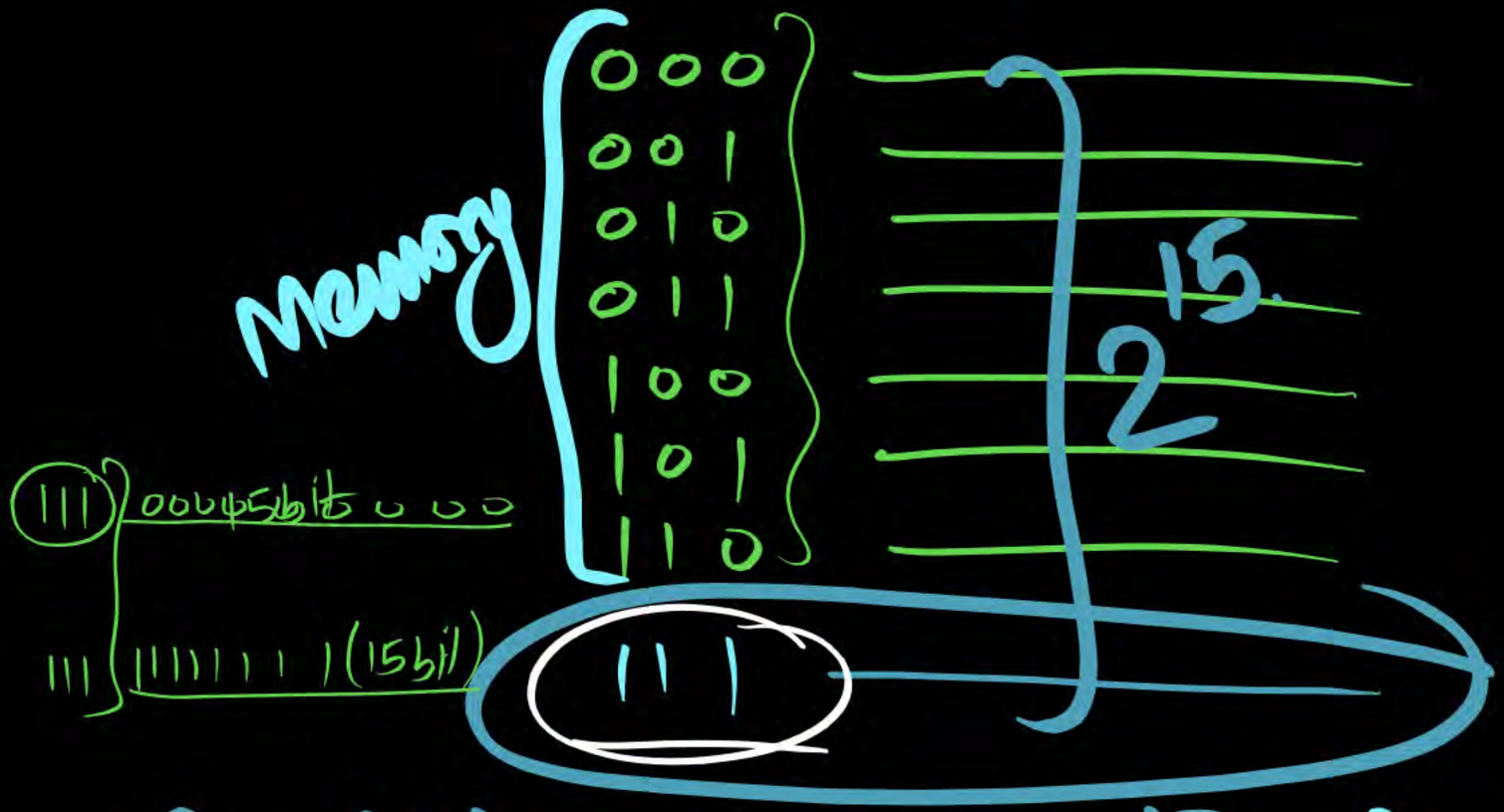
256 kByte $\Rightarrow 2^{18}$ Byte

So Address = 18 bit

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

3 MSB bit

15 bit (2^{15})



Memory Address $= 7 \times 2^{15}$ Ans

Port Address $= 1 \times 2^{15}$ Ans

THANK - YOU