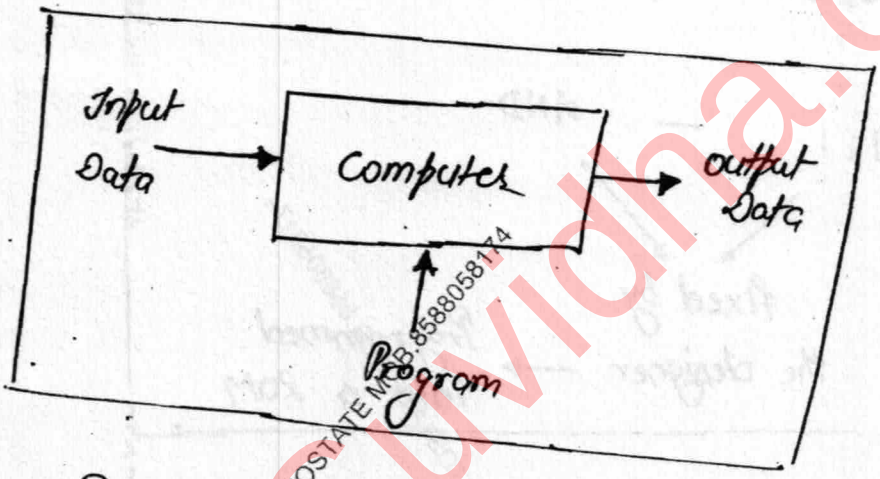
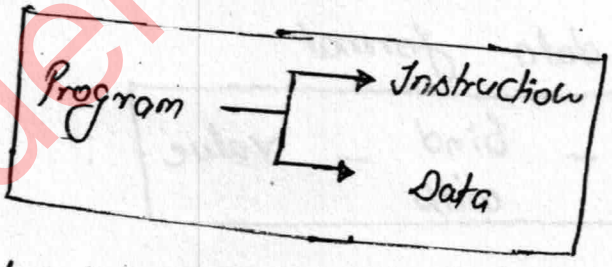
module : 1 :- Computer Functionality (1)

:- Computer : It is a computational device used to process the data under the control of a program, therefore computer system functionality is a Program execution.



:- Program : Program is a sequence of instructions along with data.



:- Instruction : It is a binary code which is designed inside the Processor to perform some task.

Ex:

Binary code	-	Bind with	-	Operation

Ex:- If X-CPU supports 8 operations (Instructions), then $\log_2 8$ bit binary code is used as opcode.

opcode	operation :
000	— +
001	— *
010	— /
⋮	
111	— AND

fixed by the designer → Programmed into a ROM

:- Data :- It is a binary code which is associated with a value based on the data format

Binary code	— bind with	— value

* Different data formats used in the computer design is as follows :

Data formats

- Fixed Point data
  - magnitude format
    - unsigned format (only +ve data)
      - n-bit range {0 to (2^n-1)}
      - 4 bit range: {0 to 15}
    - Signed-magnitude format (Both +ve, -ve data)
      - MSB | | | LSB
      - sign | value
      - {$-(2^{n-1}-1)$ to $+(2^{n-1}-1)$}
      - {-7 to +7}
  - Compliment format
    - 1's compliment format (Both +ve, -ve data)
      - {$-(2^{n-1}-1)$ to $+(2^{n-1}-1)$}
      - {-7 to +7}
    - 2's compliment format (Both +ve & -ve data)
      - {$-(2^{n-1})$ to $+(2^{n-1}-1)$}
      - {-8 to +7}
- Floating Point Data
  - Single Precision (32 bit) format
  - Double Precision (64 bit format)

Binary					
0000	-0	+0	redundancy +0	+0	No redundancy
0001	-1	+1	+1	+1	
0010	-2	+2	+2	+2	2's Compliment
⋮	⋮	⋮	⋮	⋮	
0111	-7	+7	+7	+7	
1000	-8	+8	-7	-8	
1001	-9	+9	-6	-7	
⋮	⋮	⋮	⋮	⋮	
1111	-15	+15	-0	-1	
		16 values	16 values		16 values

* General purpose computers are (8085 & 8086) designed with a fixed point ALU. So all the operations are performed on a fixed point data format.

* 4-bit data code is:

4 bit Binary code	unsigned Data	Sign magnitude Data	1's Compliment Data	2's Compliment Data
0000	0	+0	+0	+0
0001	1	+1	+1	+1
0010	2	+2	+2	+2
0011	3	+3	+3	+3
0100	4	+4	+4	+4
0101	5	+5	+5	+5
0110	6	+6	+6	+6
0111	7	+7	+7	+7
1000	8	-0	-7	-8
1001	9	-1	-6	-7
1010	10	-2	-5	-6
1011	11	-3	-4	-5
1100	12	-4	-3	-4
1101	13	-5	-2	-3
1110	14	-6	-1	-2
1111	15	-7	-0	-1
		Not in use	Not in use	

1's compliment code:

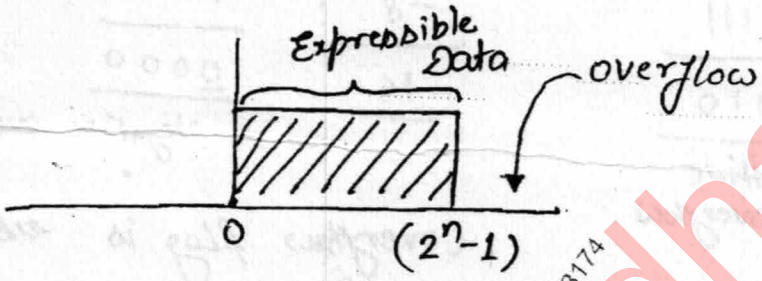
1000 : 0111 (7)
(-7)
1001 : 0110 (6)

2's compliment code:

1000 : 0111
(-8) 1
1000 = 8

1001 : 0110
(-7) +1
0111 = 7

:- "unsigned Data" :-



Note :- Carry flag is used in the CPU design, to indicate the range exceeding condition of a unsigned arithmatic.

Condition : Is there an extra bit out of MSB

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

Ex:

5 → 0101
6 → 0110
11 1011
cy = 0 cy = 0

PSW: Program Status word

Justify:

PSW
- Acc (Accumulator)
- flag reg. → cy → 0

Ex 2:

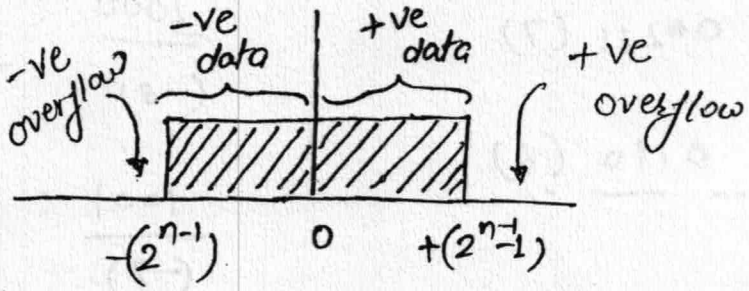
8
+9
17

1000
1001
0001
cy → 1

Ans: 1 0001

Ans: (1011)

:- "Signed Data" :- (2's compliment data).



+7	:	0111	−8	:	1000
+7	:	0111	−8	:	1000
+14		1110	−16		0000
		Positive overflow			Negative overflow

(Overflow flag is used in signed data.)

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

Condition :- There is a carry into the MSB and No carry out of the MSB (or) vice-versa when it is true flag is ~~set~~ :

→ F = set = 1 = OV

→ F = reset = 0 = NV

operation

Incarry	outcarry [w.r.t. MSB]
0	0
0	1
1	0 → 1 (OV) → 0 (NV)
1	1

("XOR")

:- Expression :-

$OV(x, y, z) : \underbrace{\bar{x}\bar{y}z}_{+ve\ "ov"} + \underbrace{xy\bar{z}}_{-ve\ "ov"}$

- MSB of operand 1
- MSB of operand 2
- MSB of Result

eg: 1 :-

+7	−	0111
+3	−	0011
+10		1010
OV = 1		OV = 1

(1) ✓ (1) ✓

Justify:

PSW
- Acc → 1010
- Reg → OV → 1 (+ve overflow)

Ans: (OV) 01010 [+10]

eg: 2:-

+7	−	0111
−3	−	1101
+4		0100
OV = 0		OV = 0

(1)(1) ✓ ✓

Justify:

PSW
- Acc → 0100
- Reg → OV → 0 (No overflow)

Ans:- 0100 (+4)

~~1100 → 0011 +1 0100~~

eg 3:-

−7	−	1001
+3		0011
−4		1100
OV = 0		OV = 0

✗ ✗

Justify :-

PSW
- Acc → 1100
- Reg → OV → 0 (No overflow)

Ans:- 1100 [−4]

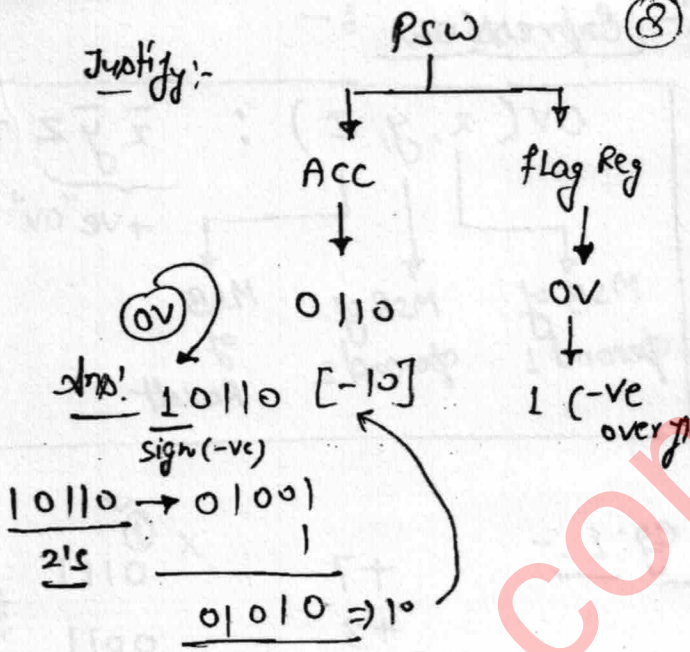
1100 → 0011 +1

eg:4:-

$$\begin{array}{r} -7 \\ -3 \\ \hline -10 \\ \hline OV=1 \end{array}$$

① × 1001 — 1101 — 0110

Justify:-



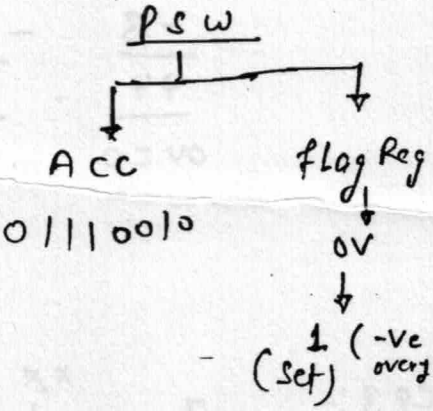
Que:- Consider the following signed binary code, perform the arithmatic addition operation. what is the status of overflow flag in the above computation:?

Data: ① 1016 1101

$$\begin{array}{r} 11060101 \\ \hline 01100010 \quad OV=1 \end{array}$$

$xy\bar{z} = OV$

$OV = 1$



Que:- Consider the following 8 bit data computation what is the status of overflow flag in computation:

(-121) + (-112)

Soln:-

$$\begin{array}{r} -121 \\ -112 \\ \hline -233 \end{array}$$

$-233 > -128$ $OV = 1$

8 bit range:

$\{-2^{8-1}$ to $+2^{8-1}-1\}$

$\{-128$ to $+127\}$

Computer Architecture :-

:- Computer System contain 3 fundamental components :-

1.) CPU (Processing unit)
2.) Memory (storage unit)
3.) I/O (External communication)

Based on the Program storage, computer Architecture is two kinds:

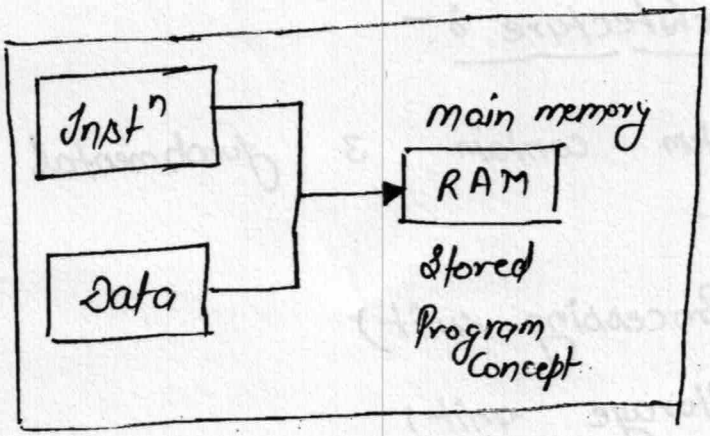
1.) Single Memory Architecture
[Von-Neumann Architecture]

2.) Multi-Memory Architecture
[Horvard Architecture]

⇒ "Von-Neumann Architecture" :-

:- In this Architecture, uses Program is always required in main memory, so CPU will be executing the main memory part of the Program.

* Main-memory is a volatile memory, so we can load the different application Programs into a main-memory. therefore CPU will be executing the different Programs in the same system.



Von Neumann

Implemented as a micro-Processor design.

eg: 8085 µP ⟶ 64KB RAM

8086 µP ⟶ 1MB RAM

used in the Personal computer Design

eg: General Purpose computers.

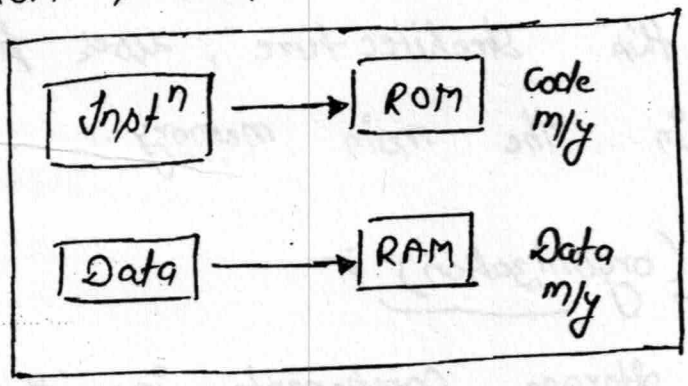
Vonn-Neumann :- (Controled flow machine)

⇒ "Horvard Architecture"

* In this Architecture, Program Logic is stored in the 'Code memory' and data is stored in the 'data memory'.
* Code memory - is a permanent memory, that is "ROM."
* Data memory - is a temporary memory, i.e. "RAM."
* In this Architecture CPU always executes the Pre-defined Program in the Rom chip on a different data elements.

Horvard architecture :-

- code m/y
- Data m/y
- ROM & RAM



★ Implemented as a micro (μ) - controller design.

eg:- 8051 μ controller → 4kB ROM, 128B RAM

★ used in the intelligent system design.

eg: ~~Intel~~ Embedded systems.

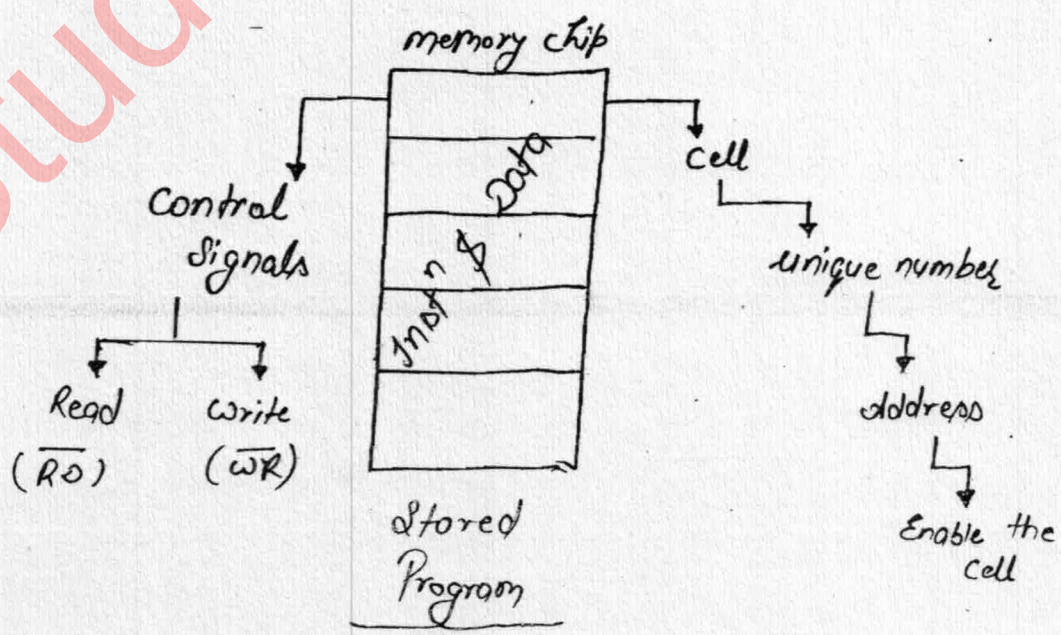
Computer organization :-

Computer organization shows the implementation of a von- Neumann architecture.

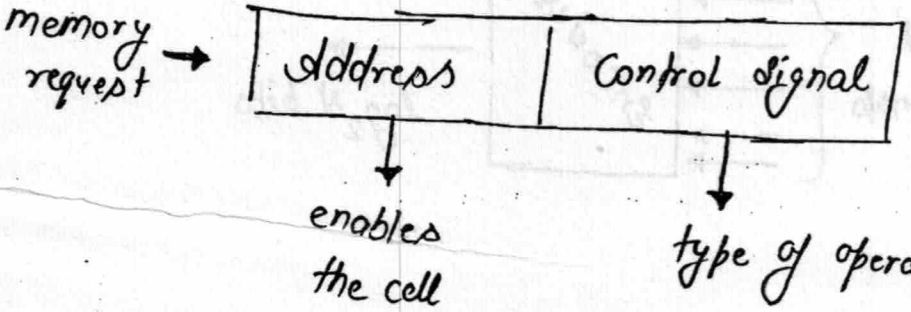
According to this Architecture, user program is present in the main memory.

:- Memory Design (organization) :-

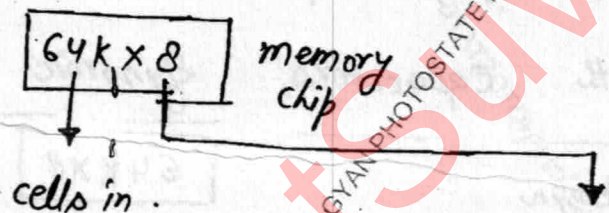
* Memory is a storage component in the computer.
* Memory chip is divided into equal parts called as cells.
* Each cell is identified with a unique number, called as address.
* memory chip recognizes the control signals to indicate the type of a operation.



★ CPU generates the memory request to access the Program from the memory chip. i.e.



★ Number of address bits required to access the memory is depends on the capacity of a memory chip. Denoted as: 64k X 8, 64k X 16, 64k X 32 .....



cells in the chip i.e. 64k cells

address bits required to access the chip i.e. $\log_2 64k = 16$ bit

Cell size { # bits stored in the cell }

Eg: m x n

{ $\log_2 m$ bit Addressesed cell holds n bit data }

Address space / Address range / memory map of a memory chip

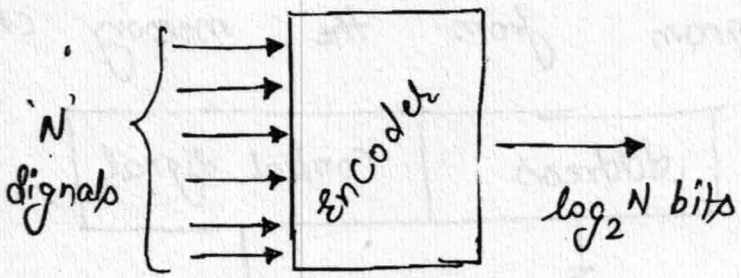
i.e. (0000 0000 0000 0000 to 1111 1111 1111 1111)$_2$

Hexa-decimal address space (0000 to FFFF)

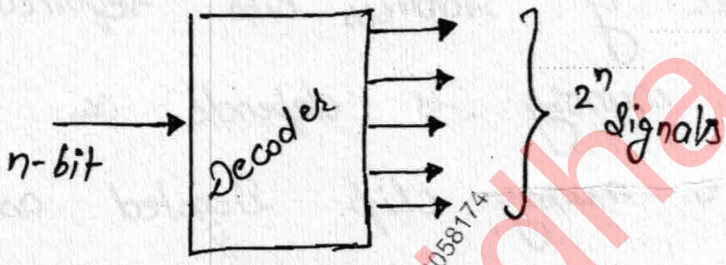
for low level languages like assembly → for Coding

Note :-

Encoding :-

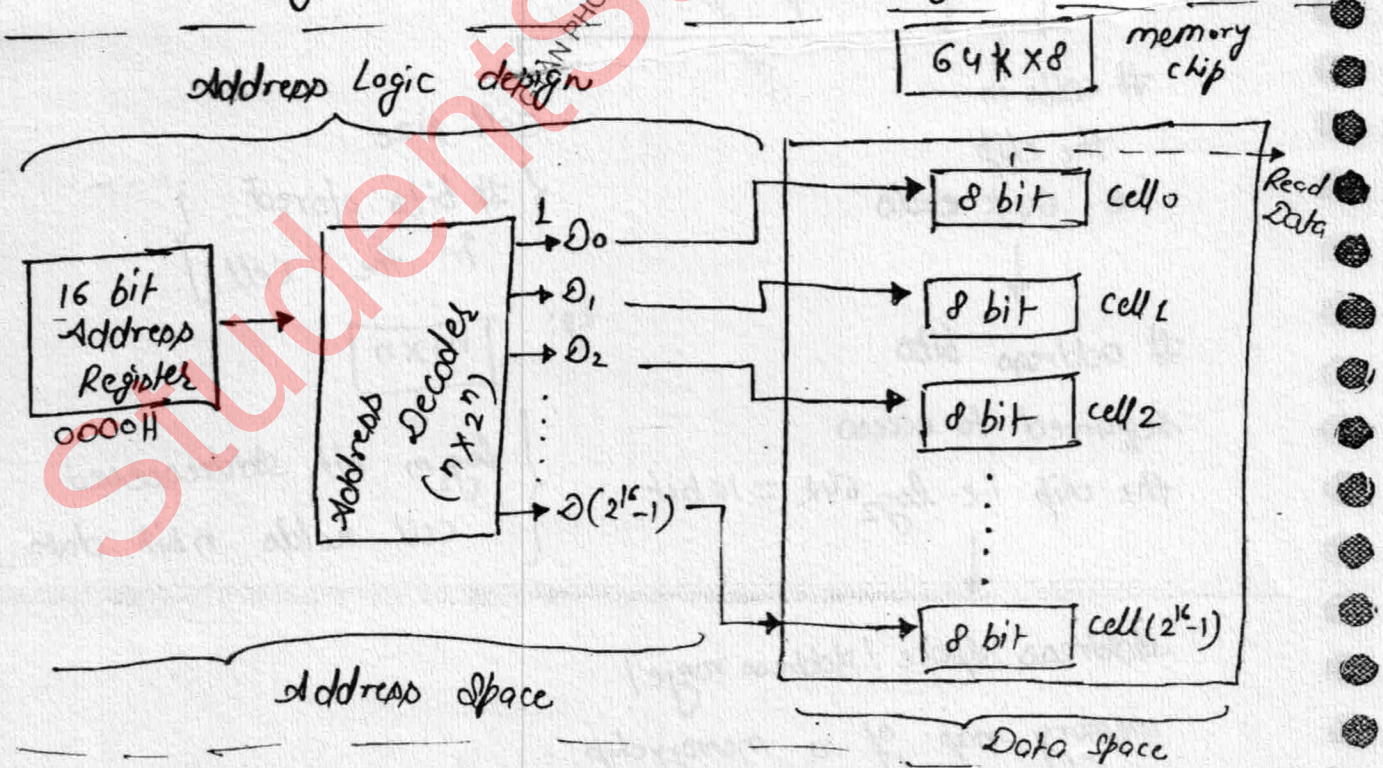


Decoding :-

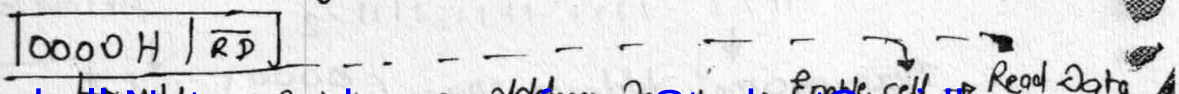


⇒ RAM, designed with flip-flops, Static RAM

⇒ RAM, designed with capacitors, Dynamic RAM.



eg: CPU generates the m/y request:



⇒ In the CPU, Binary data is referred with the following ~~for~~ referencial elements :-

CPU [Processor]	Bit	Nibble	Byte (B)	word (ω) (word size = word length → # bits Processed at a time.)
1 cell ← 8 bit up	1 bit	4 bit	8 bit	8 bit
2 cell ← 16 bit up	1 bit	4 bit	8 bit	16 bit
4 cell ← 32 bit up	1 bit	4 bit	8 bit	32 bit
n bit up	1 bit	4 bit	8 bit	n bit

:- Byte addressible Vs word addressible memories :-

1.) when the memory cell size is 8 bit then corresponding address space is called as byte address.

Eg: 4×8 : 2 bit Addressed cell holds 8 bit data
64×8 : 6 bit Address "
256×8 : 8 bit Addressed cell holds 8 bit data
$1k \times 8$: 10 bit "
$2^n \times 8$: n bit "

Cell size must be of 8 bit

→ Byte addresses (cell size)

* When the memory cell size is designed with a word length of a CPU then the corresponding address space is called as word address.

eg:
- 4 x w : 2 bit Addressed cell holding word size data
- 64 x w : 6 bit "
- 256 x w : 8 bit "
- 1k x w : 10 bit "
- 2^n x w : n bit "

(w: cell size must be a word; n bit: word addresses)

Note :-

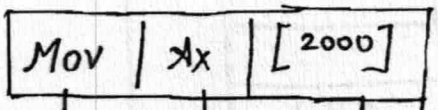
1.) Default memory configuration in the computer design is Byte addressible so in the memory, Data is stored in a Byte wise sequence

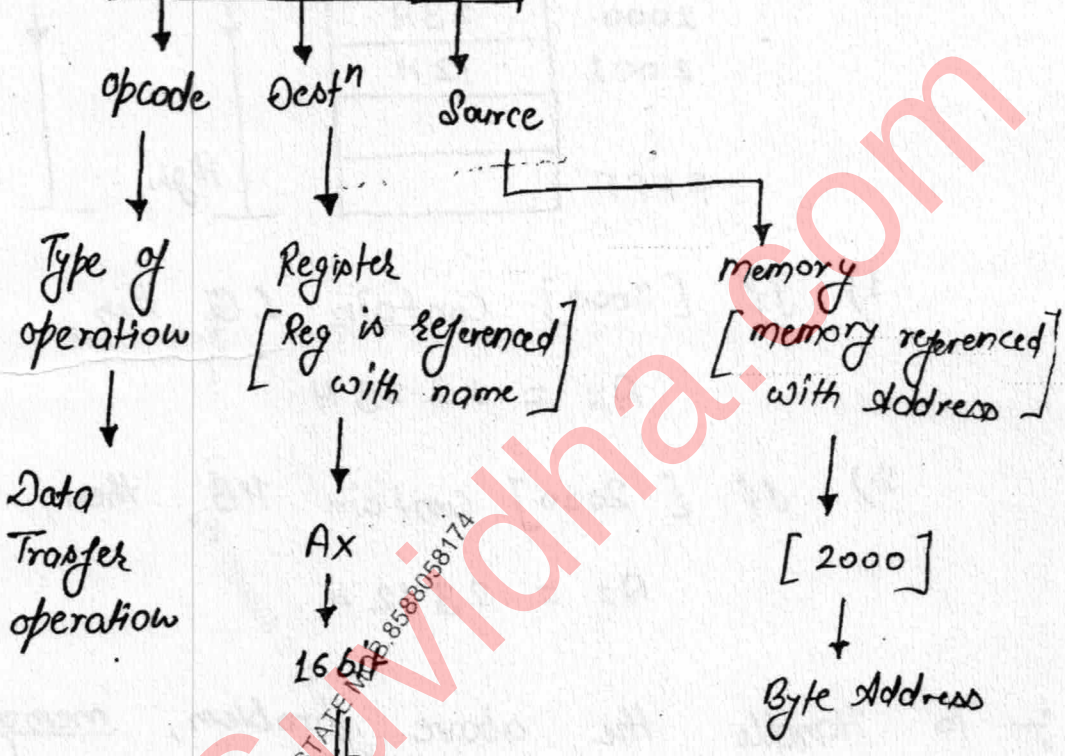
2.) In the CPU, operations are performed on a word formats so based on the word length, memory interfacing will be ~~existed~~ adjusted to access the data word by word from the memory chip.

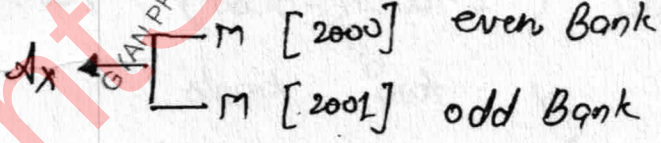
Date 09/08/2017

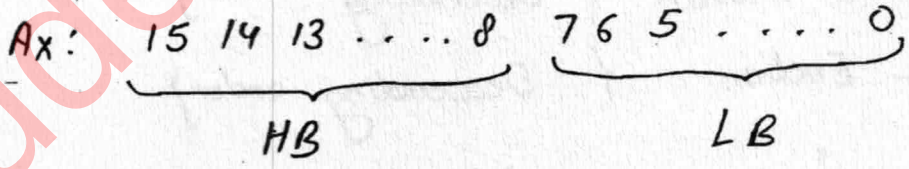
eg: 8086up (16 bit CPU)

Instruction:



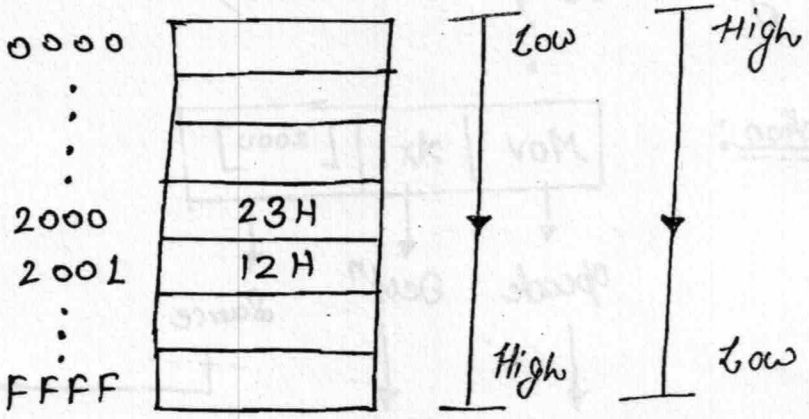






:- During the Execution of a above instruction two possible outcomes are generated due to lack of a order of data storage in the memory. i.e.

ie memory contents :



1) If [2000] contain LB then,

AX = 1223H

2.) If [2000] contain HB then,

AX = 2312H

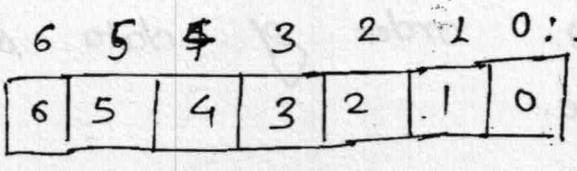
:- To Handle the above problem, memory address interpretation ('Endianness') technique is used.

It is of two kinds :

① Little-Endian { Ascending order }

② Big-Endian { Descending order }

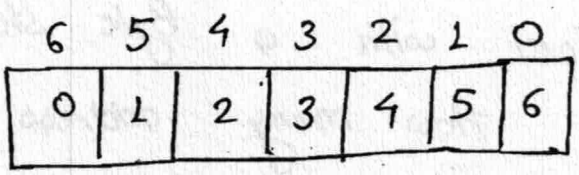
* In the 'little endian,' Lower address contain lower byte and Higher address contain higher byte



Lower Address : contain LB & " Little Endian "

Higher Address : contain HB.

★ In 'Big - Endian' Lower address contain higher byte and Higher address contain Lower byte.

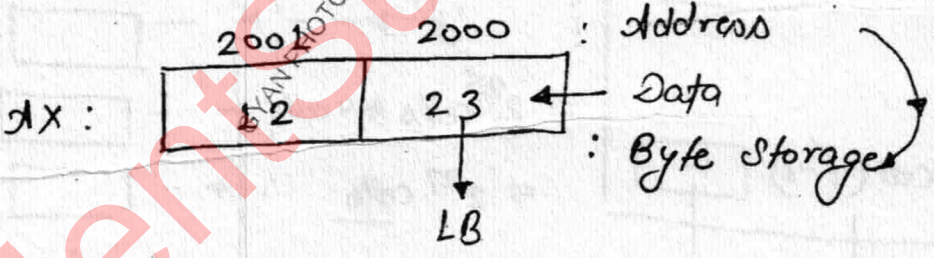


Lower Address : contain HB
Higher Address : Contain LB

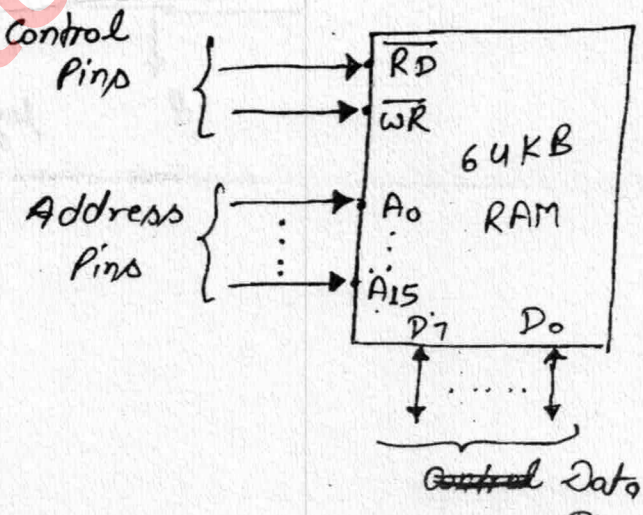
"Big Endian"

⇒ Default address interpretation technique in the computer design is "little - endian" therefore above instruction generates the following output :

Little - Endion :



AX : 1223H



Que:- Consider a hypothetical processor which supports 32 MW memory. word length (W) of the CPU is 32 bit. Processor is enhanced with a Byte addressible memory space. How many address bits are required to address the new memory chip.

$32 \times M \times 4B = 128MB$

$= 27$ bits

System old: