Computer Organization

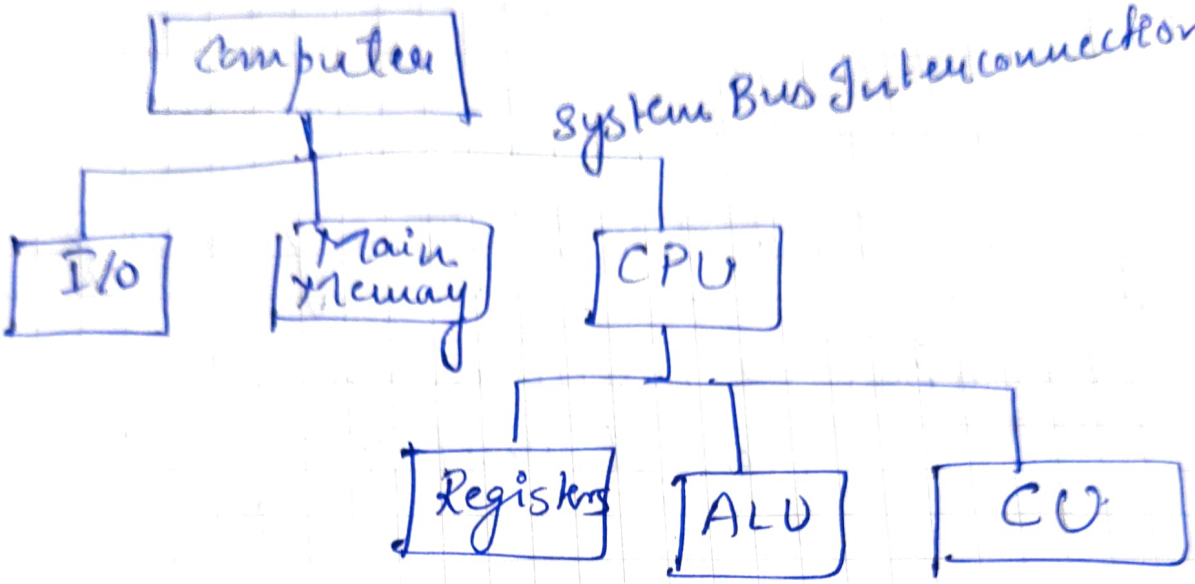
1. Structure and behaviour of a computer system as seen by the user
2. It deals with the components level of a connection in a system
3. Explain exact arrangement of units in the system is and their interconnection
4. Expresses the realization of architecture

An organization is done on the basis of architecture

Computer Architecture

1. Deals with the way hardware components are connected together to form a computer system
2. Acts as a interconnection between hardware & software
3. It provides understanding of the functionalities at system level
4. Architecture expresses in terms of instructions, addressing modes & registers
5. While designing a computer system architecture is considered first

Basic organization of computer and Block level Description of Functional Units.



CPU: handles processing and control signals of the computer

Main Memory: It is used to store program & data

Input/Output: Communicate between computer & external environment

System Interconnection:

CPU, Memory, I/O are connected through conducting wires called as system bus

CPU structure

Control Unit → Controls all the components by sending out control signals.

ALU (Airthematic & Logic Unit)

All airthematic & Logical operations are carried out

Registers: Used for temperary storage.

Internal Bus → Communication b/w. Control Unit & registers

Control Unit Structure Level 3

* Implemented using microprogrammed approach.
* Executes micro instructions which governs the functionality of the control Unit.

Von Neuman Architecture

① what is stored program concept in digital computer (3)

② explain von Neuman architecture in details ;

Stored program Concept

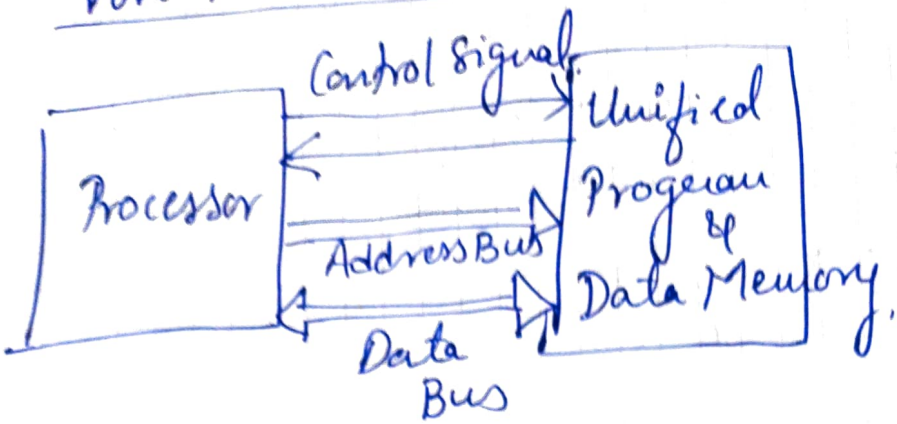
* It is the idea that the instructions. i.e, program & data to be stored together in a shared memory in binary form, in order to perform a variety of tasks in sequence or intermittently.

* Program is then fetched sequentially one inst at a time from the memory & then executed using Fetch, Decode – Execute

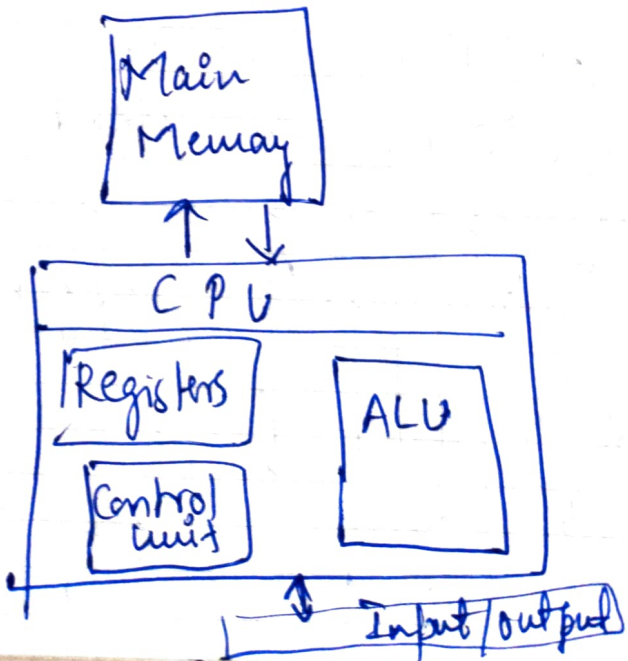
* storing a program be electronically is that the instructions can could be modified by the computer as determined by intermediate computational resu

* Accumulator is used for short term. intermediate storage of AL data in computers CPU's.

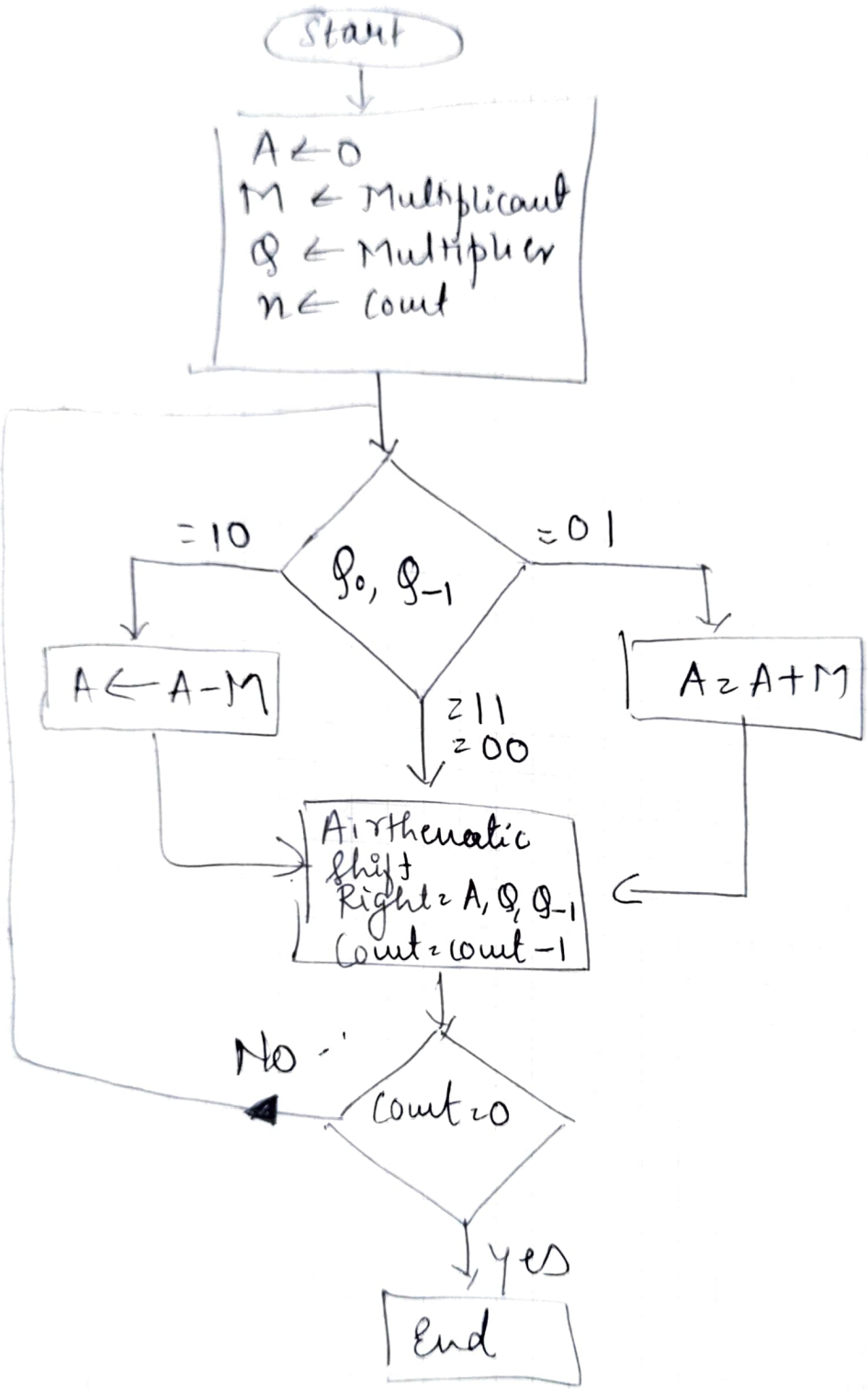
Von Neumann architecture



→ Named After Mathematician John Von Neumann
→ Works on the Concept stored program.
→ 1940 at Princeton.
→ Relies on 3 Concepts
- Unified memory for data & program storage
- Sequential execution of program
- Data is stored in memory with unique address location.



Booth's Algorithm



M Q

Multiply (−5) and 2

M = −5 = 1011

Q = 2 = 0010

n	A	Q	Q_{-1}	Action
4	0000	0010	0	Initialization
3	0000	0001	0	Shift Arithmetic
	0101			10 so A = A − M
3	0101	0001	0	Shift
2	0010	1000	1	
	1101			A = A + M
2	1101	1000	1	Shift
1	1110	1100	0	
1	1111	0110	0	

Result

1 1 1 1 0 1 1 0 (−10)

↓ 1's comp

0 0 0 0 1 0 0 1

\+ 1 ← 2's com

0 0 0 0 1 0 1 0 (10)

A = 0000 − 14

A 00000
−M +10010
10010

n	A	Q	q_{-1}	Action
5	00000	11011	0	Initialization
4	10010	11011	0	A − M
	11001	01101	1	shift
3	11100	10110	1	only shift
2	01010	10110	1	A + M
	00101	01011	0	shift
1	10111	01011	0	A − M
	11011	10101	1	shift
0	11101	11010	1	shift

A2 A1 A0
1 1 1 0 0
0 1 1 1 0
1 0 1 0 0

00101
10010
10111

11101101010

↓

$(-9) \times (6)$

$M = -9 \rightarrow 01111$

$-M = 01001$

$01001 + 10110 = 10111$

$01001 + 10110 + 1 = 10111$

n	A	Q	q-1	Action
5	00000	00110	0	Initialization
4	00000	00011	0	only shift
3	01001	00011	0	A-M. 00000
	00100	10001	1	shift 00011
				01001
2	00010	01000	1	only shift
1	11001	01000	1	A+M 00010
	11100	10100	0	shift 10011
				11001
0	11110	01010	0	only shift

11110 01010 (-54)

(-7×4)

-7×4

-7 (M) | 1001 (M)

7 (-M) | 0111 (-M)

00111

11000

+ 1

01001

Q = 00100

n	A	Q	Q_{n-1}	Action
4	00000	0100	0	Initial
3	0000	0010	0	only shift rt
2	0000	0001	0	only shift rt
1	0111	0001	0	A-M 0000
	0011	1000	1	shift +01001
				0111
0	1100	1000	1	
	1110	0100	0	A+M 0011
				shift +1001
				1100

11100100 (-28)

-3×-7

Restoring division

(1) Load Register A with 0's & with dividend.

(2) shift A, Q by 1 position (left)

(3) Subtrat A from M

(4) If MSB of (A) = 1 then $Q_0 = 0$ and Restore
(A) = 0 then $Q_0 = 1$ No Restore.

(5) Repeat till bit positions

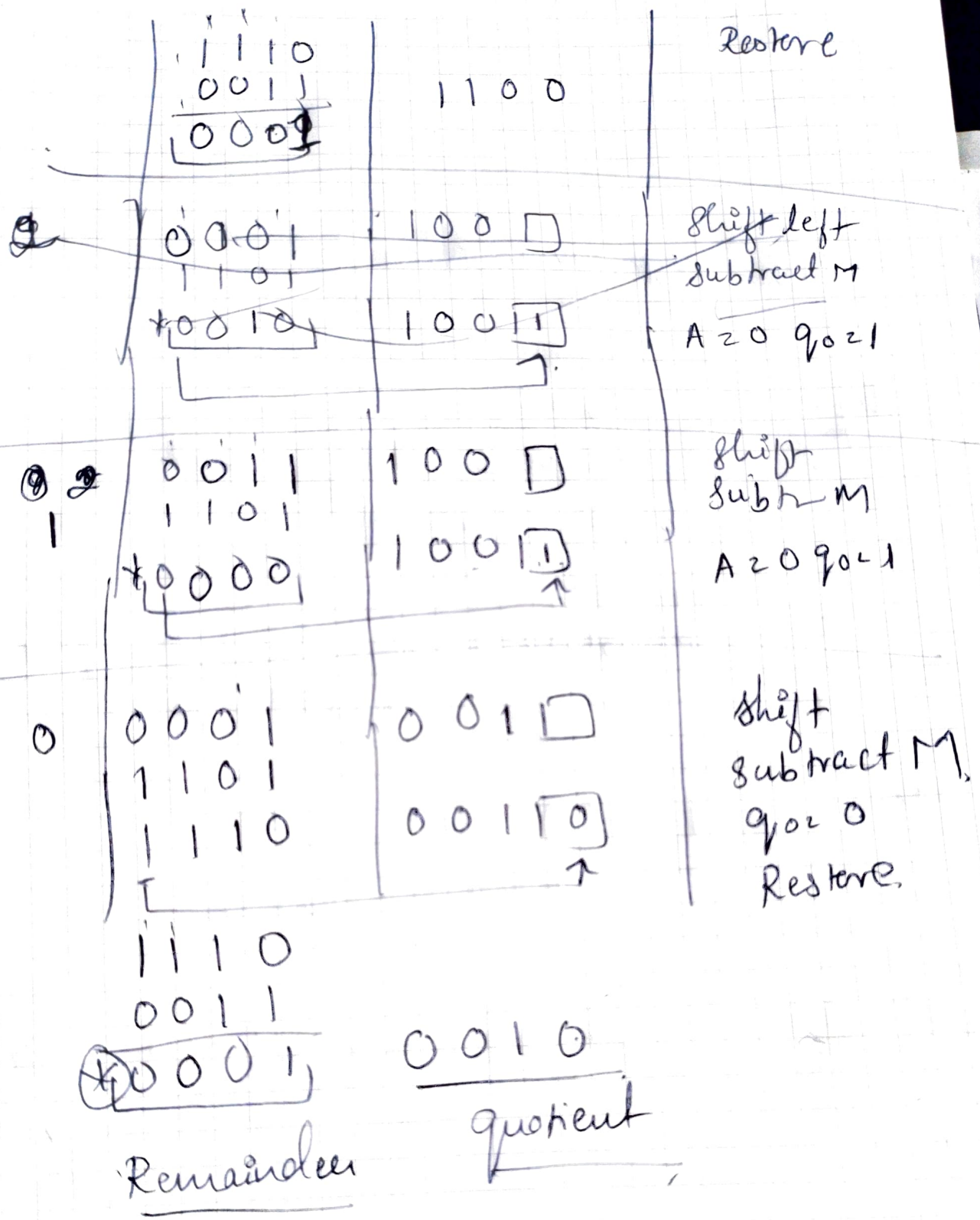
7 ÷ 3

M = 3 divisor
Q = 7 → (3)(7) → divident
0 0 1 1

M = 0011
−M = 1101

1100
+1
1101

n	A	Q	Action
4	0000	0111	Initial.
3	0000 1101	1110	Subtract M
	1101	1110	A = 1 $q_0 = 0$
	0011		Restore.
	0000	1110	
2	0001 1101	110	Shift left Subtract M
	1110	1100	A = 1 $q_0 = 0$


1 1 1 0
0 0 1 1
0 0 0 1
1 1 0 0
Restore
0 0 0 1
1 1 0 1
+ 0 0 1 0
1 0 0
1 0 0 1
Shift left
Subtract M
A < 0 q0 = 1
0 0 1 1
1 1 0 1
+ 0 0 0 0
1 0 0
1 0 0 1
Shift
Subtract M
A < 0 q0 = 1
0 0 0 1
1 1 0 1
1 1 1 0
0 0 1
0 0 1 0
Shift
Subtract M.
q0 = 0
Restore.
1 1 1 0
0 0 1 1
+ 0 0 0 1
0 0 1 0
Remainder
quotient

① 8 ÷ 2.

② 7 ÷ 4

Non-Restoring Division

Previous

Step 1 If MSB(A) = 0 shift A & Q register 1 bit left and Subtract M from A

Previous
MSB(A) = 1 shift A & Q to left & Add M to A

Step 2 if MSB(A) = 0 $q_0 = 1$
MSB(A) = 1 $q_0 = 0$

last (After n steps) if MSB of (A) = 1 then. Restore add M to A.

1100
+ 1

5 ÷ 3

3 | 5

M = 3, Q = 5

M_2 0011
$-M$ 1101

Q 0101 → 1010
+1
$-Q$ 1011

n	A	Q	Action
4	0000	0101	Initial
	0000	1011	Shift left
3	1101	1010	SubA−M
	1011	0101	Shift rt
2	1110	0100	Add.
	1110	0100	Set q0
	1100	1000	shift
1	1111	1000	add
	1111	1001	set q0
	0010	0001	shift
0	0010	0001	add
	0010	0001	set q0

1011 + 0011 = 1110

1100 + 0011 = 1111

1111 + 0011 = 0010

0010 Remainder

0001 Quotient

Floating Point

The standard specifies interchange and airthematic formats and methods for binary and decimal floating point airthematic in computer progeraming.

Single Precision

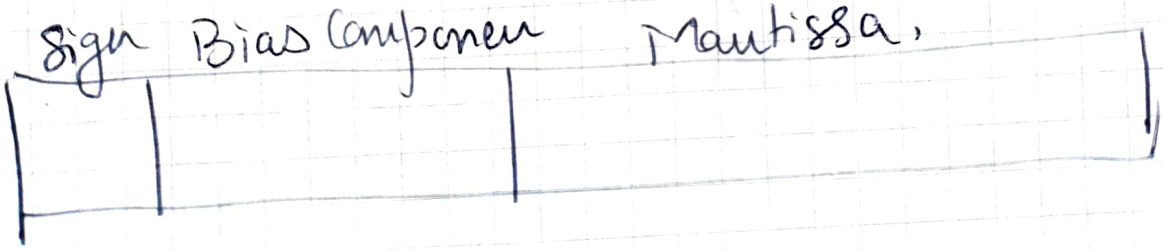
→ Using Binary Base, the least significant 23 bits from D_0 to D_{22} are used to store the fractional part i.e Mantissa.

→ D_{23} to D_{30} → used to store 8 bit exponent.

→ D31 bit is sign bit.

Eg $(0.15625)_{10}$ → binary $(0.00101)_2$.

$0.00101 = \boxed{1.01_2 \times 2^{-3}}$



Bias Component

$-3+127 = 124$

01111100 — 8 bit

Sign bit = 0

fraction = 1.01

↓

01 and rest all 0's

0	01111100	01000000000000000000000

Double Precision — 64 bit and Bias component add 1023

Bias

$-3+1023 = 1020$

01111111100

Sign 0

0	01111111100	01000 ... all 0's

~~84.25~~

Double precision

① D_0 to D_{51} are used to store fractional part

② D_{52} to D_{62} 11 bits Bias Component

③ D_{63} Sign bit

① 34.25

$(00100010.01)_2$

1.00010001×2^5

Mantissa 00010001000000000000000000000000000000

$127+5 = 132 \Rightarrow 10000100$

0	10000100	00010001000 ...

8 bit

127.125

↓

Binary 1111111.001

$1.1111111001 \times 2^{6}$

Mantissa

11111001 all 0's

Single Precision

6 + 127

133

10000101

Double Precision

6 + 1023

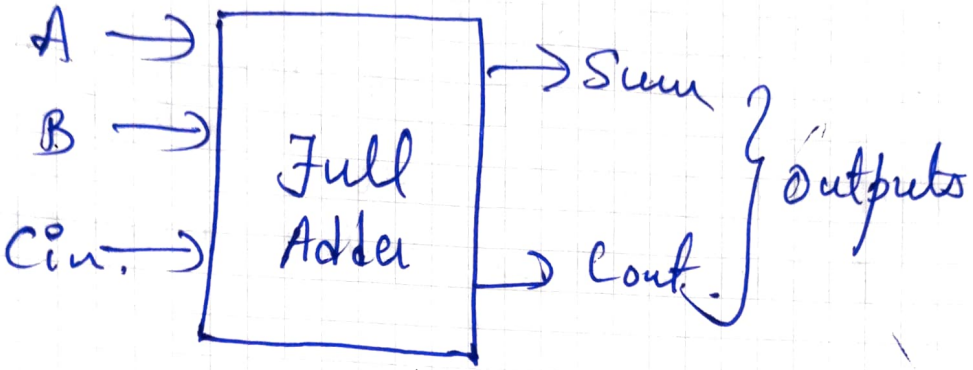
1029

10000000101

Full adder

1) Adds 3 inputs and 2 outputs.
A & B are inputs & 3rd input is Cin

2)



Carry

$Cout = \bar{A}BCin + A\bar{B}Cin + AB\bar{C}in + ABCin$

$= \bar{A}BCin + A\bar{B}Cin + AB(\bar{C}in + Cin)$

$= \bar{A}BCin + A\bar{B}Cin + AB$

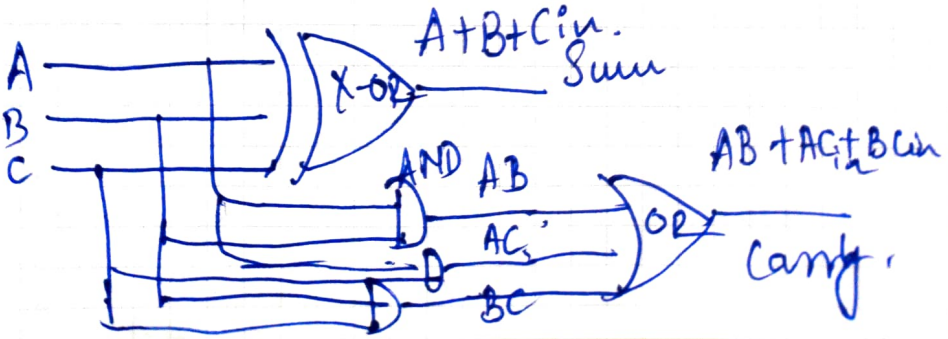
$\bar{A}BCin + ABC + A\bar{B}Cin + ABC + AB$ Adding ABC + ABs

~~$AB + BCin + ACin$~~

$(\bar{A} + A)BCin + (\bar{B} + B)ACin + AB$

$BCin + ACin + AB.$

Carry $\boxed{AB + ACin + BCin}$ rewriting



Decoder

→ It is a Multiple input & Multiple output device.

→ Decoder is a combinational circuit that converts n lines → input.

2^n → output (Max output can be less)

→ Applications of decoder are

Converting binary code to other codes

Eg Binary to octal.
Binary to Hex.
Binary decimal.

Octal Base is 8 → 0 to 7

Hence input needs to be given is 3

output 2^3 so 3×8 decoder.

Hex Base is 16 → 0 to 16

4 input (2^4) output

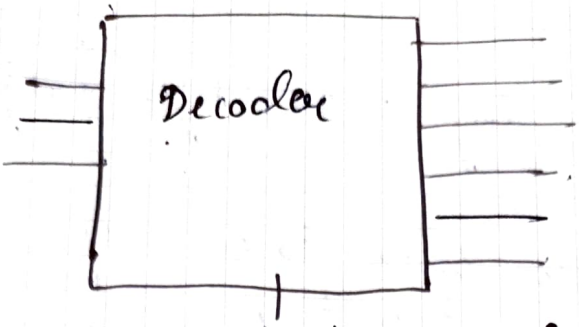
4×16 decoder.

Dec Base is 10

4 input

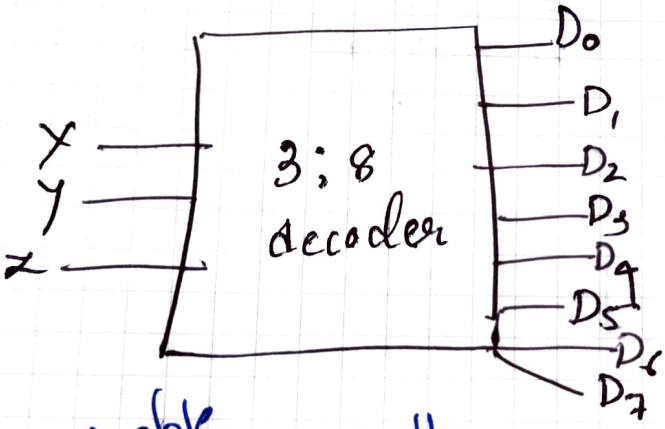
2^4 output but base is 10 so we write 10

4×10 decoder



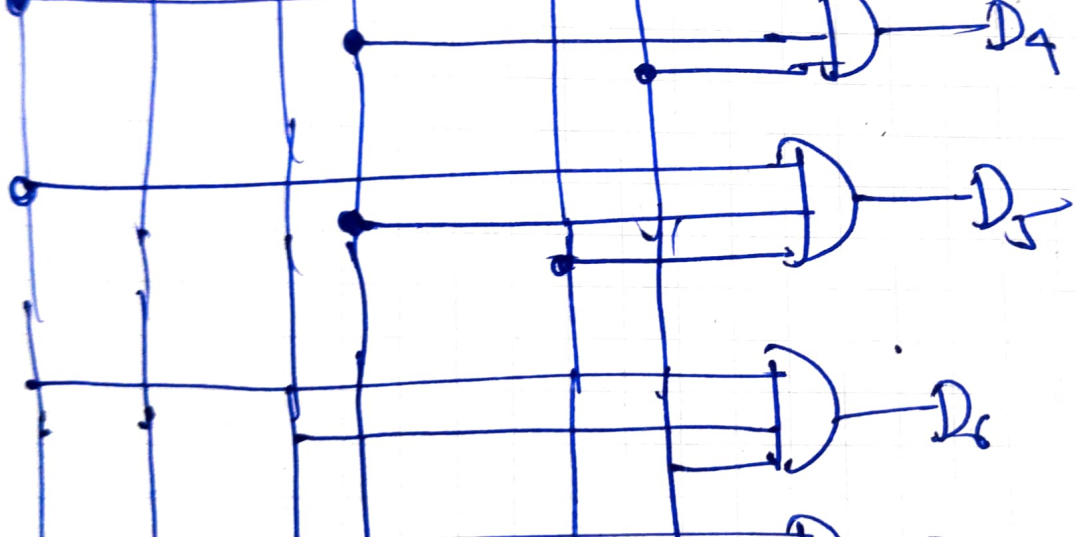
Enable → circuit enabled then it works.

3x8 decoder.



enable

X	Y	Z	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0


D_4
D_5
D_6

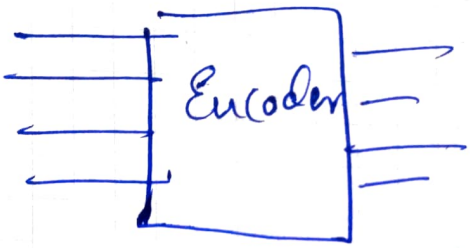
Encoders

→ Multiple input multiple output device

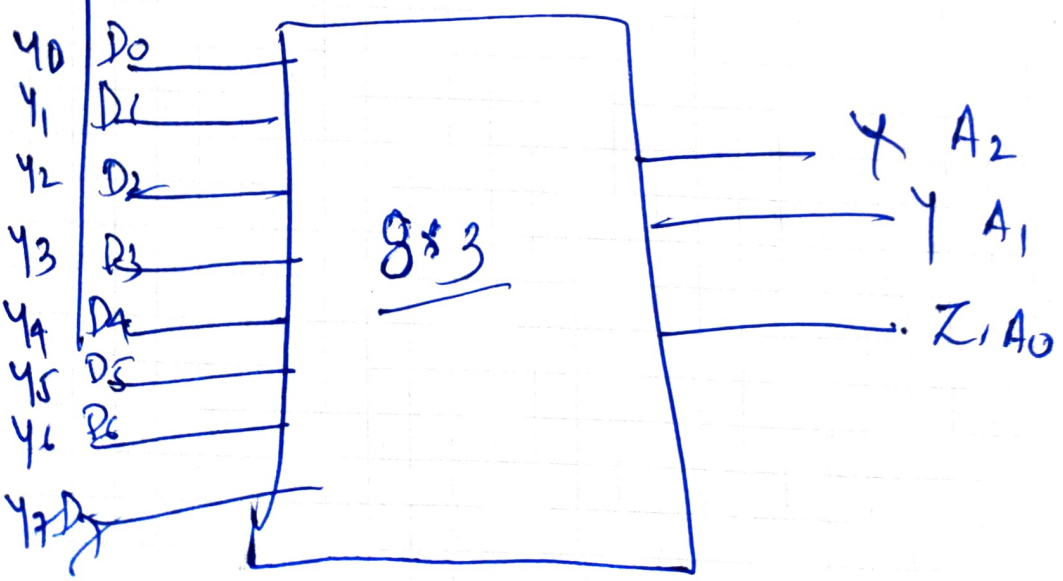
→ It converts Octal to Binary.
Hex to Binary
Dec to Binary

→ Opposite to Decoders

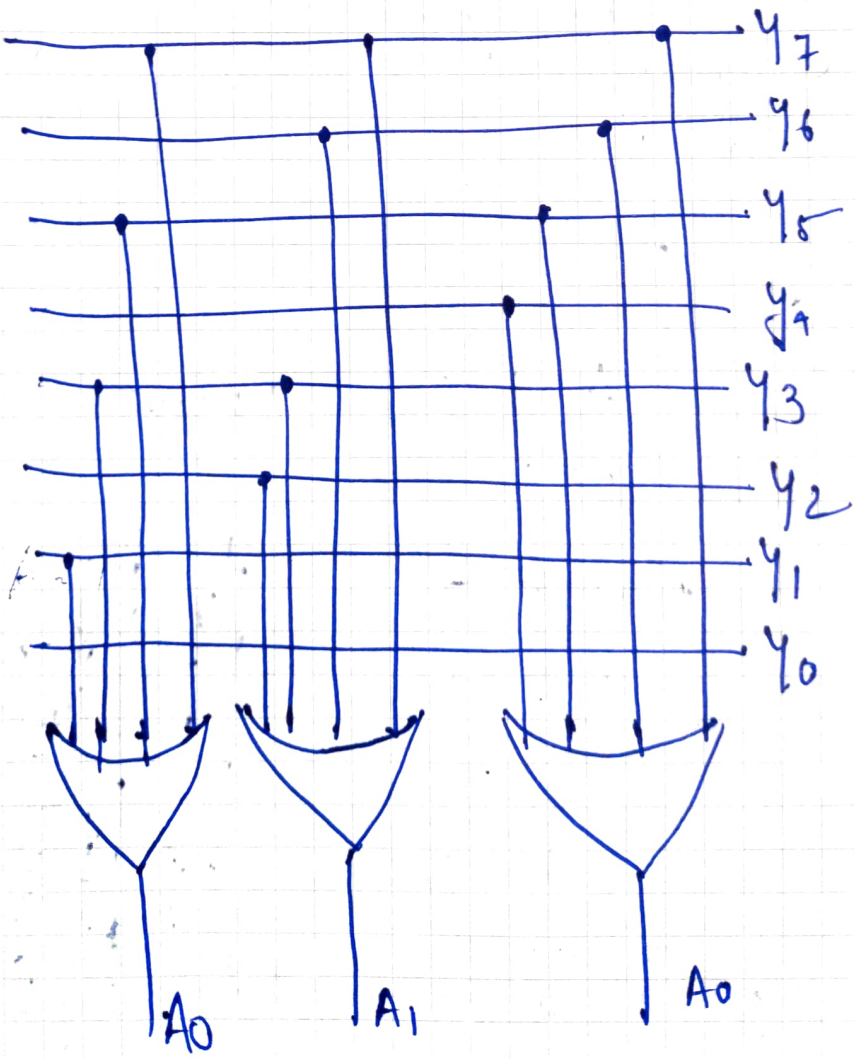
Here we have 2^n input
n output



8:3 or 8×3 Encoder.



y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	A_2	A_1	A_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1



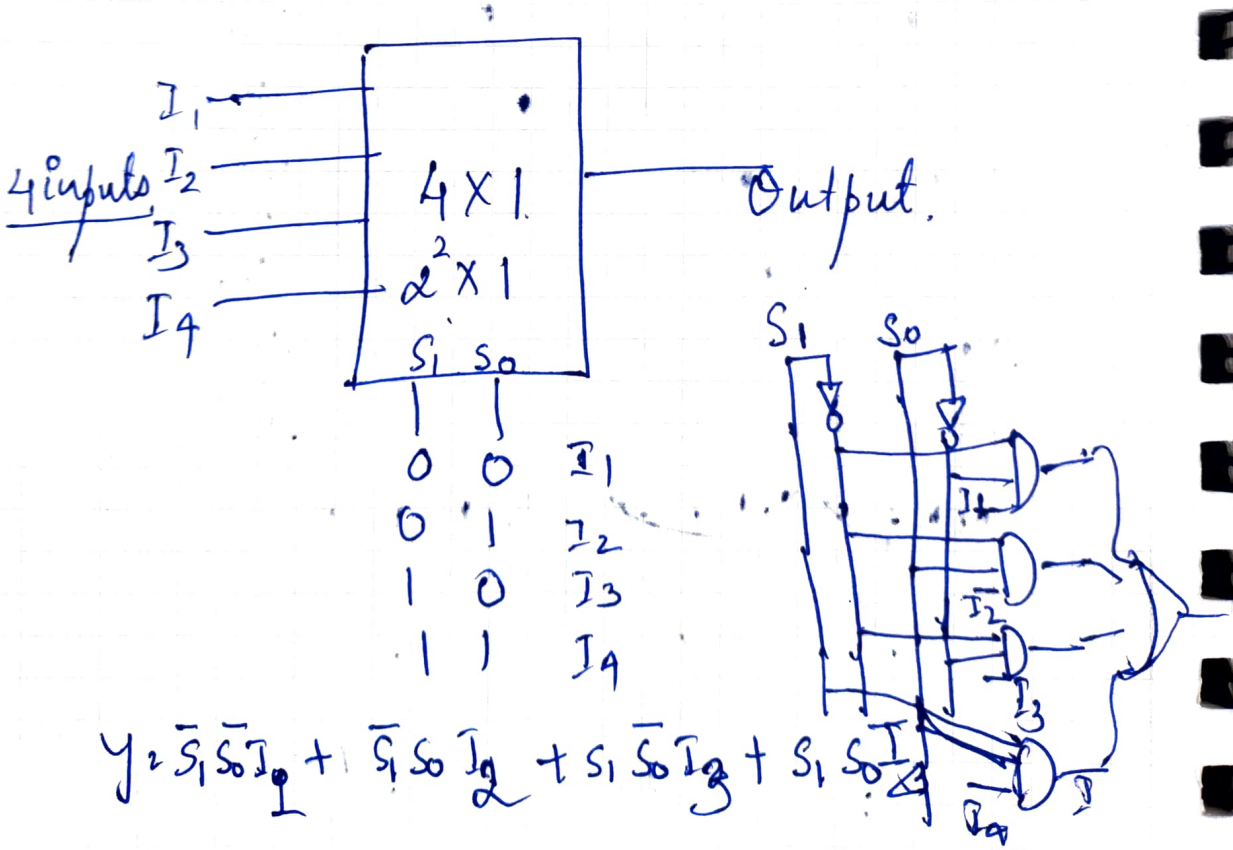
Multiplexer

→ It is a combinational circuit that has 2^n input lines and single output

$2^n : 1$

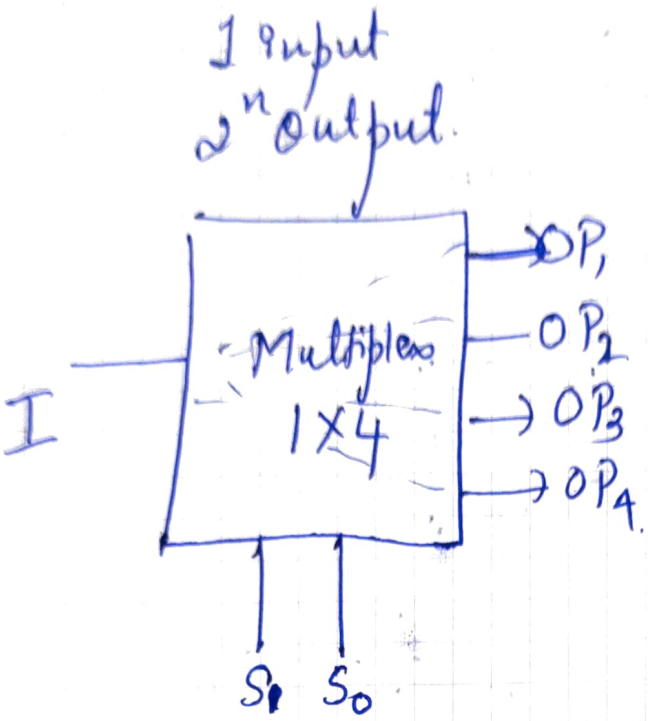
→ Multiplexer is an electronic switch that connects 1 out of n inputs to an output

→ It is a functionally complete. That is all Boolean functions can be realised using one multiplexer without any other gates.



$Y = \bar{S}_1 \bar{S}_0 I_0 + \bar{S}_1 S_0 I_1 + S_1 \bar{S}_0 I_2 + S_1 S_0 I_3$

Demultiplexer.



1 : 2^n selecting line

input Propogates

S_1	S_0	OP_1	OP_2	OP_3	OP_4
0	0	I	0	0	0
0	1	0	I	0	0
1	0	0	0	I	0
1	1	0	0	0	I

Uses

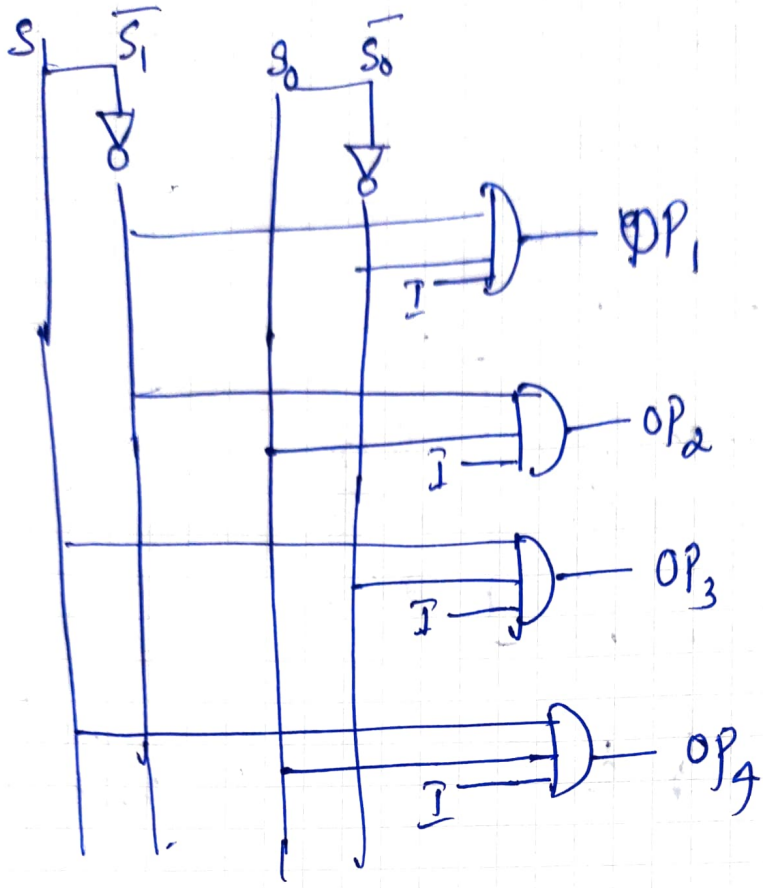
To send signals in network.
multiplexing & demultiplexing

$OP_1 = \bar{S}_1 \bar{S}_0 I$

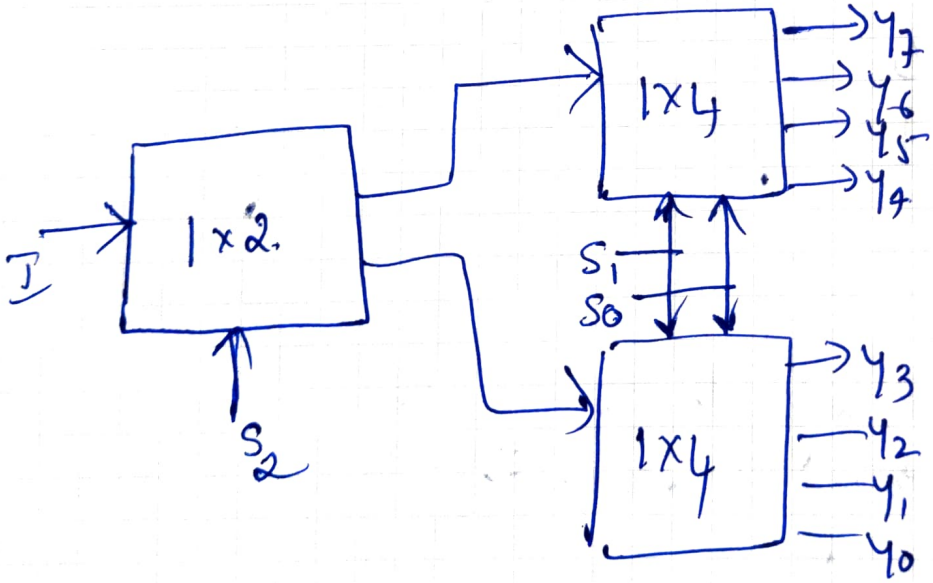
$OP_2 = \bar{S}_1 S_0 I$

$OP_3 = S_1 \bar{S}_0 I$

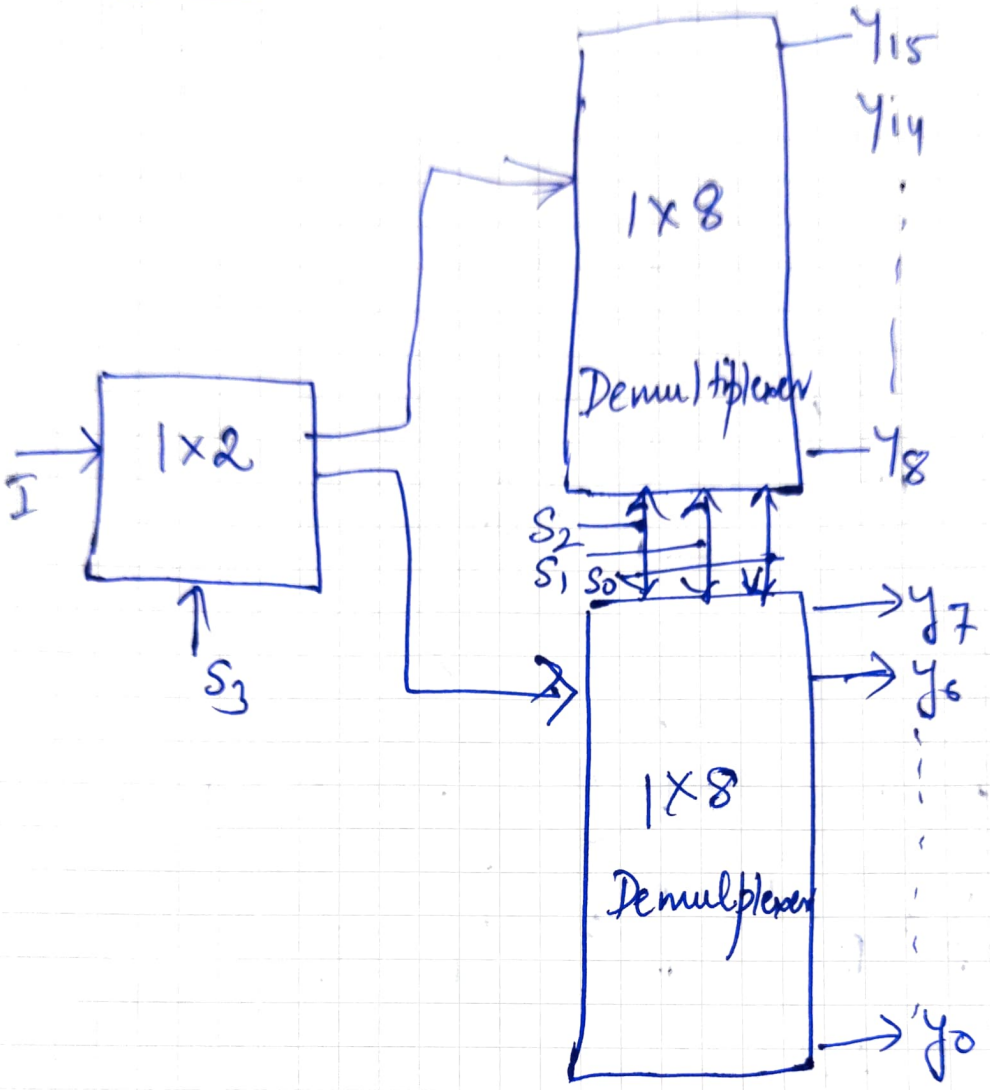
$OP_4 = S_1 S_0 I$



1X8 demultiplexer

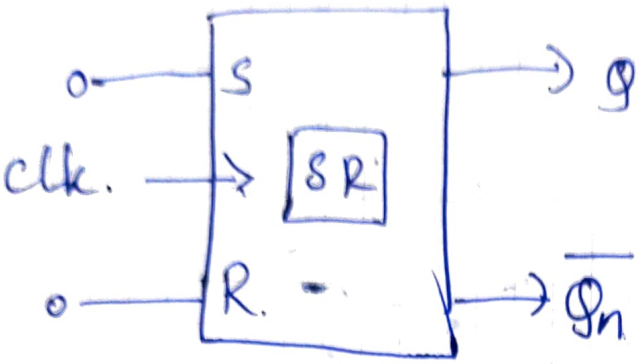


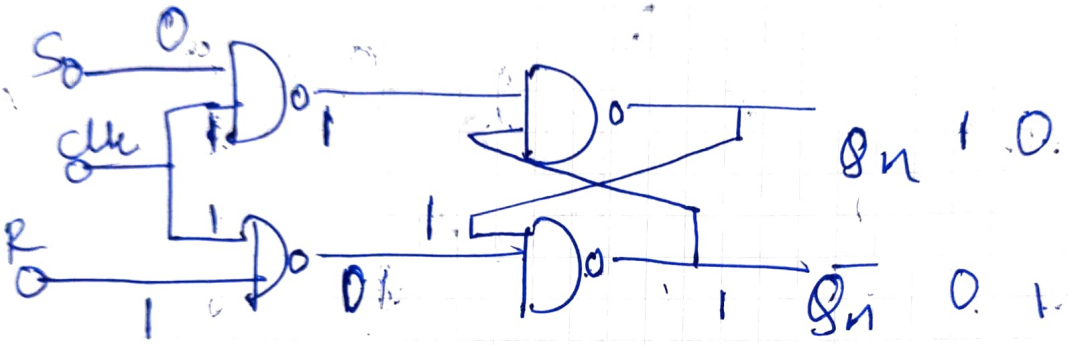
1X16


1X8
Demultiplexer
1X2
I
S3
S2
S1
S0
1X8
Demultiplexer
y15
y14
y8
y7
y6
y0

Flipflop

S-R





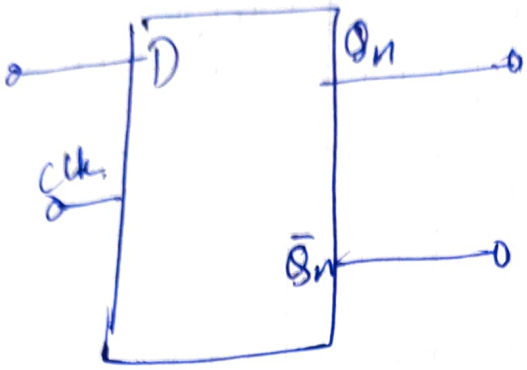
Truth table

S	R	Q_n	Q_{n+1}	State
0	0	0	0	
0	0	1	1	
0	1	0	0	
1	0	0	0	
1	0	1	1	
1	1	0	1	
1	1	1		

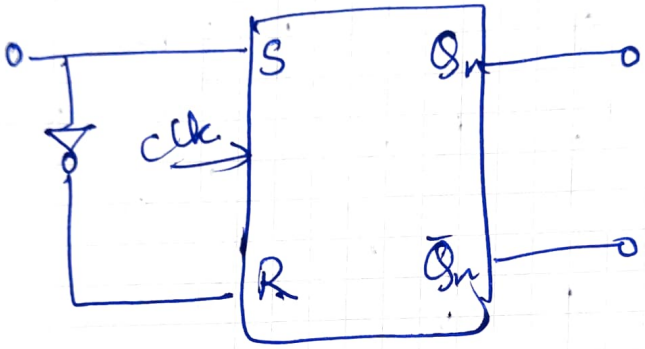
S	R	Q_n	Q_{n+1}	State
0	0	0	0	HOLD
0	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	0	0	1	set
1	0	1	1	
1	1	0	X	invalid
1	1	1	X	

S	R	Q_{n+1}	State
0	0	Q_n	HOLD
0	1	0	Reset
1	0	1	set
1	1	X	invalid

D flip-flop

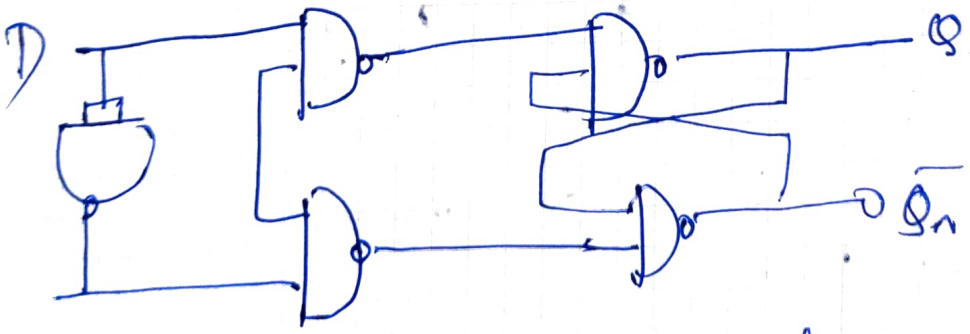


D	Q_{n+1}
0	1
1	1



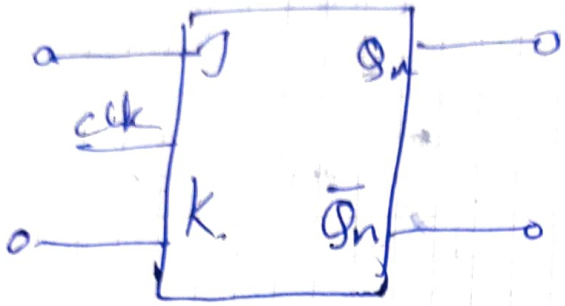
$S = D$

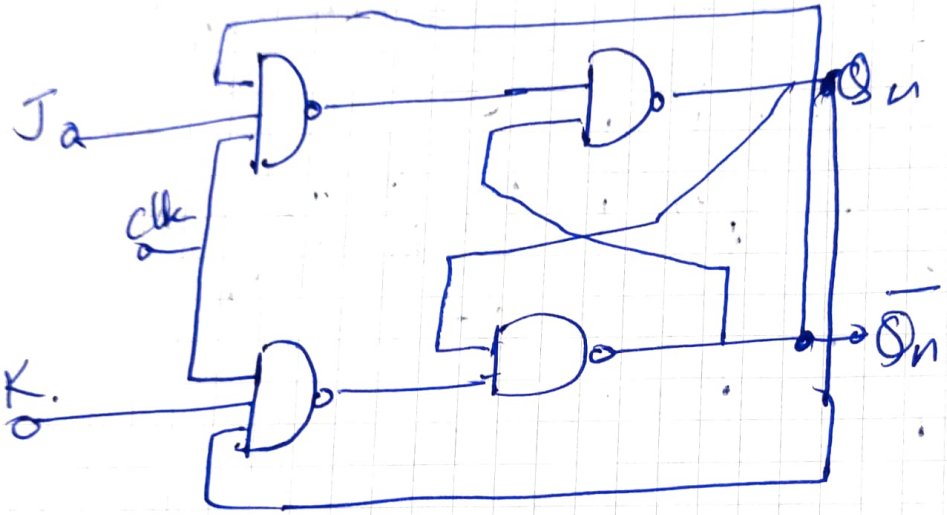
$R = \bar{D}$



D	Q_n	Q_{n+1}	
0	0	0	State Nochage
0	1	0	
1	0	1	Nochage
1	1	1	

JK flip flop



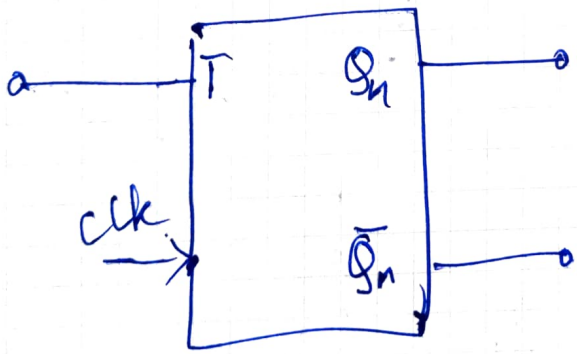


$S = \overline{J\bar{Q}_n}$ $\quad R = \overline{KQ_n}$

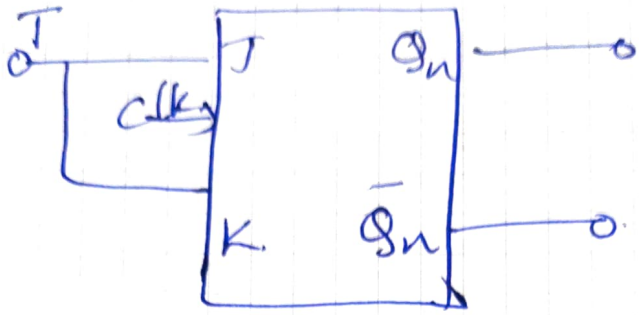
J	K	Q_{n+1}	
0	0	Q_n	No change (Hold)
0	1	0	Reset
1	0	1	set
1	1	$\bar{Q}_n$	Toggle.

J	K	Q_n	Q_{n+1}	State
0	0	0	0	No change
0	0	1	1	
1	0	0	1	Set
1	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	1	0	1	Toggle
1	1	1	0	

T flip flop



It is a single ~~input~~ input version of JK flip flop



T	Q_n	Q_{n+1}	
0	0	0	Q_n
0	1	1	
1	0	1	$\bar{Q}_n$
1	1	0	

Truth table

T	Q_{n+1}
0	Q_n
1	$\bar{Q}_n$

State diagram

Register Organisation

(a) User Visible Registers

These registers are used by the processor and programmer for minimizing the Usage of Memory.

(I) General purpose Register:

These are used to hold operand for opcode, These are used for addressing functions.

(II) Data and Address registers

(a) Data Registers:
Use to hold explicitly the data used for processing

(b) Address Register: includes segment registers, Index registers and stack pointers

(c) Segment Registers:
Hold segment base address in segmented addressing.

Index registers

Use to hold index address in indexing addressing modes.

② Control and status Registers

ⓐ Program Counter (PC)

It holds the address of the next instruction to be executed.

ⓑ IR (instruction register)

Holds the instruction most recently fetched from the memory location

ⓒ MAR (Memory address register)

Holds address of memory location

ⓓ MDR (Memory data register.
or MBR (Memory buffer "))

Holds the data to be written to memory or ~~holds~~ most recently read from memory

The instruction cycle

(a) Instruction address calculation (IAC)

→ Address of the next instruction is computed by adding fixed number to the previous instruction address

→ The required instruction address calculations are carried out.

(b) Instruction Fetch

→ The calculated instruction address is sent on the external Address bus and from the memory system the instruction is fetched. from into the processor or CPU.

→ Then stored in Instruction register (IR)

(c) Instruction operation decoding

Here the instruction is decoded means it determines the type of operation to be performed and operands to be used.

(d) Operand address calculation (OAC)

The address of the operand is computed that is to be fetched from memory or I/O space.

→ This stage is the function of addressing mode being used by the inst and the translations of address needed by virtual memory system.

(c) Operand fetch (OF)

→ Here operand is fetched from the memory or is read from an I/O device by sending the operand address computed in the previous stage, over the address bus and receiving operand over the data bus.

→ The operand could be register a memory location or an I/O port.

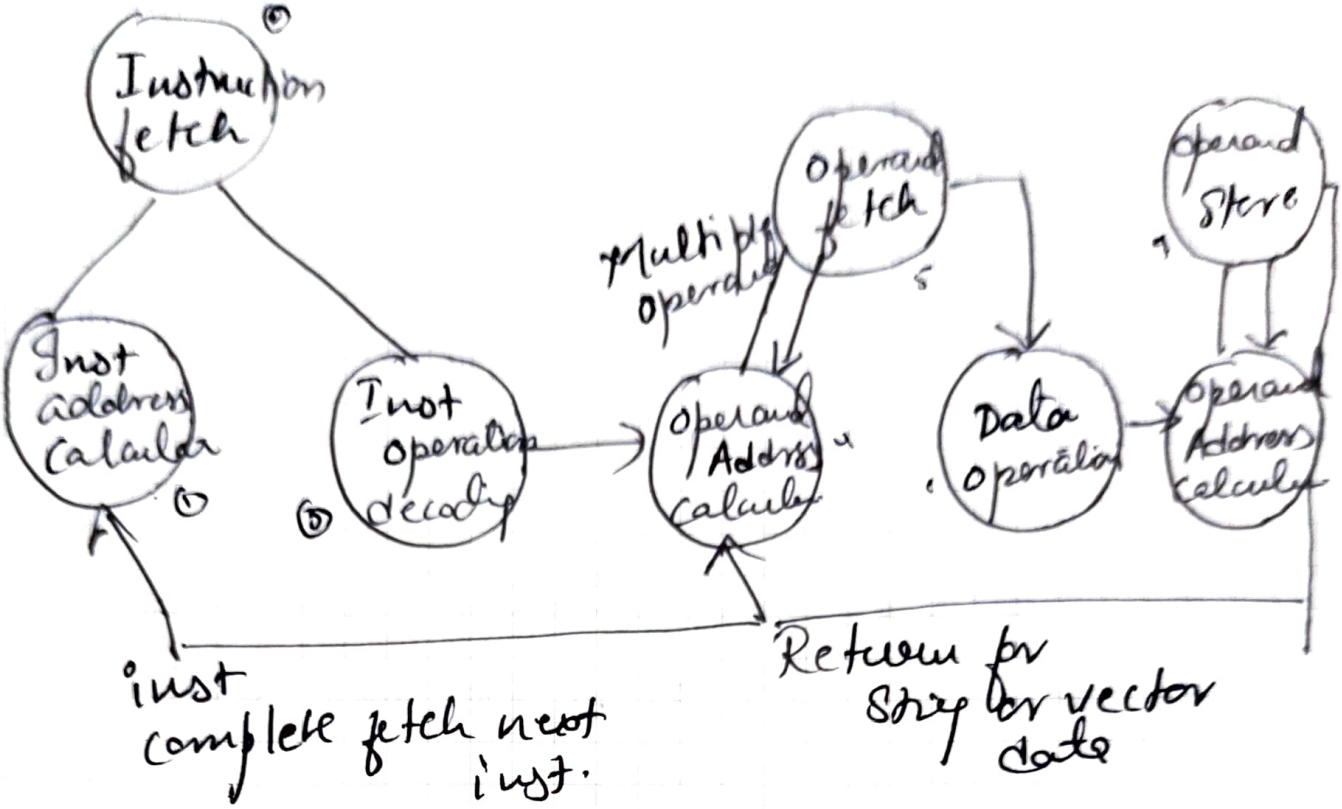
(d) Data operation

→ Performs operation indicated in the instruction

→ Here inst currently under processing is actually executed

(e) operand store

write the result back to memory.



Instruction Interpretation & Sequencing

→ Instructions are fetched from the program memory and they are sent to the Instruction decoder.

→ In instruction decoder, the instructions are decoded and their meaning & functionality is understood.

→ This is then used for generating the different control signals internally over the CPU as well as externally on the system bus.

→ This operation of decoding and understanding the instruction is called as Instruction Interpretation

→ Based on the interpretation next steps are taken in order to execute that instruction

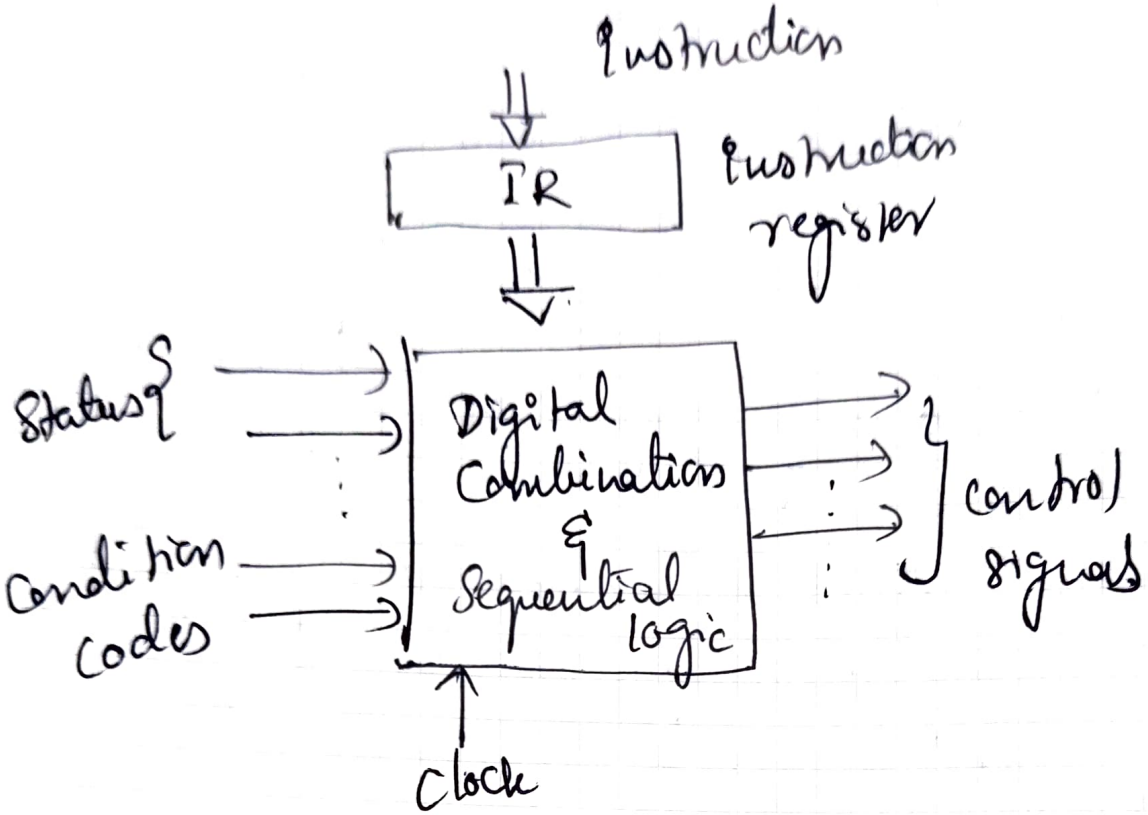
Instruction Sequencing

* Once the instruction are decoded and interpreted, they are sequenced for the execution.
* Firstly the control signal

Module 4: Control Unit Design

Hard wired Control Unit

→ It is a mechanism of producing control signals.
→ Generated using Designed as a sequential logic circuit
→ Final circuit is constructed by physically connecting the components such as gates, flip, flops. Hence the name.
→ It consists of a circuit that outputs desired controls for decoding & encoding functions.
→ Instruction is loaded in IR. is decoded by the instruction decoder.
→ Inputs to the encoder are given from the instruction step counter decoded, external inputs & condition codes
→ All these inputs are used and individual control signals are generated
→ The end signal is generated after all the inst get executed.



(1) State tabel Method

T-States	Instructions			
	I_1	I_2	....	I_N
T_1	C_{11}	C_{12}	...	C_{1N}
T_2	C_{21}	C_{22}		C_{2N}
⋮				
T_N	C_{M1}	C_{M2}	...	C_{MN}

- Here the behaviour of control unit is represented in the form of a table, which is called as state table
- Here, each row represents the T-states & the columns represent instructions
- Every inst of the specific column to each row indicates which control signal will be produced in the corresponding T-state of an inst.
- Here the hardware circuitry is designed for each column (i.e for a specific inst) for producing control signals in different T-states.

Drawback :-

In modern processors, there is a very large number of instruction set.

∴ The circuit becomes complicated to design difficult to debug and if we make any modifications to the state table then large part of the circuit need to be changed.

- There are many redundancies in circuit design like the control signals are required for fetching the instruction is common and which is repeated for N number of inst.
So cost of circuity may increase

Delay element Method

→ control unit behaviour is represented in the form of a flow-chart.

→ Each step represents a control signal that needs to be produced for processing the instruction

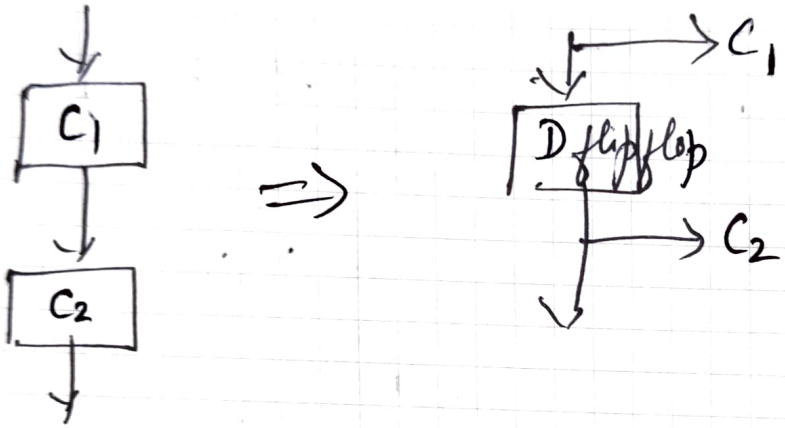
→ If all steps of the instruction are performed this means the inst is executed Completely

→ Control signals performs micro-operations and each micro-operation requires one T-states.

→ For every micro-operations which are Endependent, they are required to be performed in different T-State.

→ Therefore for every consecutive control signal an exactly 1-state delay is required which can be produced with the help of DFF.

→ ∴ D flip-flops are inserted between every 2 consecutive control signals.



→ The D FF is introduced between each pair of control signals ∴ after a control signal is generated, then the delay element before that control signal is not in use until before the next just required that control signal.

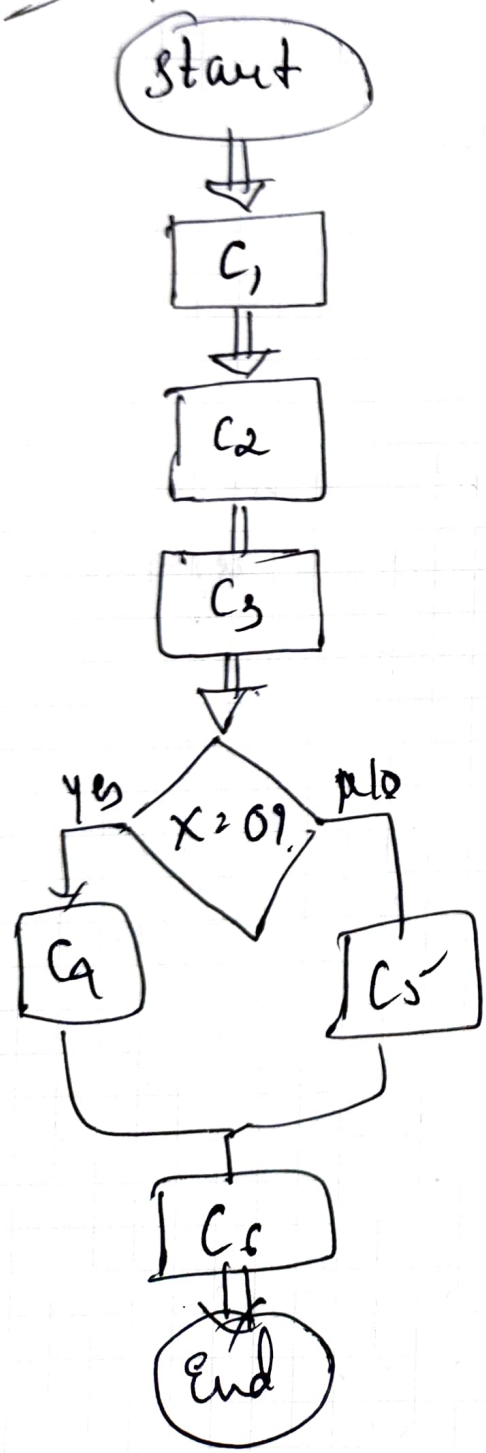
→ ∴ All D flip-flops, only one will be active at a time.

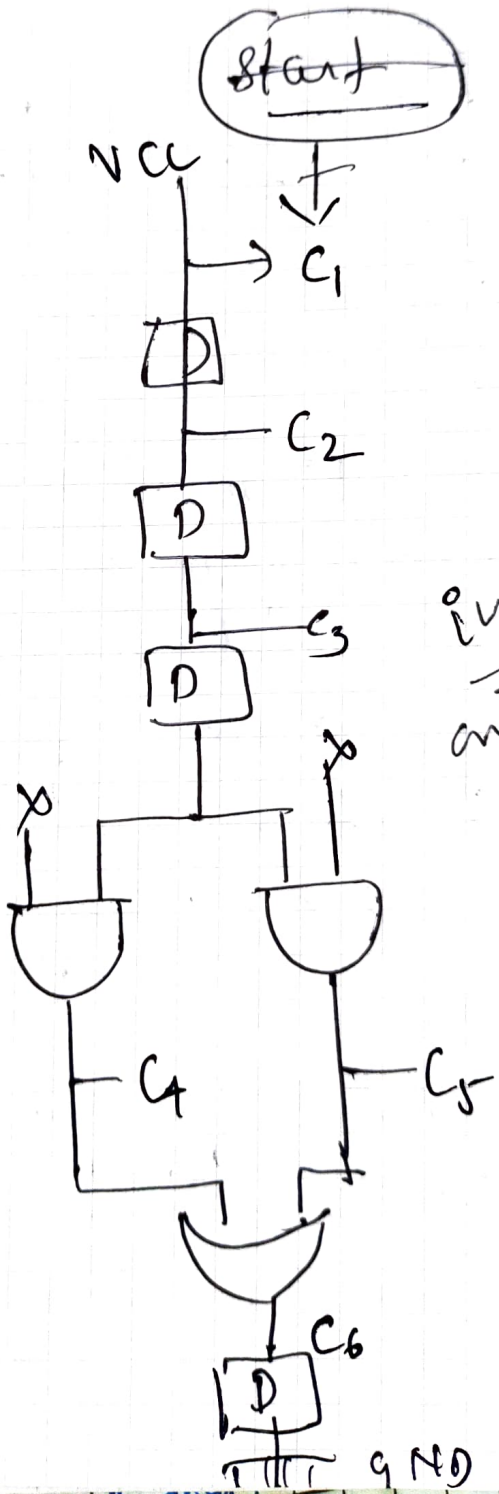
∴ this method is called one hot method.

* To combine all → OR gate
(multiple entry point)

* for decision box → AND gate

Example





independent only D flip flop

Advantage

→ logic approch so reduce circuit complexity

→ for common signals which need to be generated in every inst for them only one circuitry can be designed.

Disadvantage

No. of inst increases the no. of D FF for generating delay is increased so overall circuit complexity & cost increases.

Microprogramed Control Unit

* It uses sequence of micro instruction in Microprograming language
* Mid way between Hardware & Software
* It generates a set of control signals
* It is easy to design, test & implement
* flexible to modify.

Terminologies

Control Signals: group of bits

Control Variables: Binary variable specify microoperation

Control Word: string of 1's & 0's represent Control Variable

Control Memory: Memory contains Control Word.

Micro Instruction: Contains control words stored in control memory specifies control signals for execution of micro operation

Microprogram → Contains sequence of micro instruction

Eg Micro Inst — Control Signal.

MAR ← R_3 — MAR_{in}, R_{3out}

Stored in Control Memory

* Implemented Using Programing language.
* Sequences is carried out by executing a program consisting of micro-instructions

Micro-program

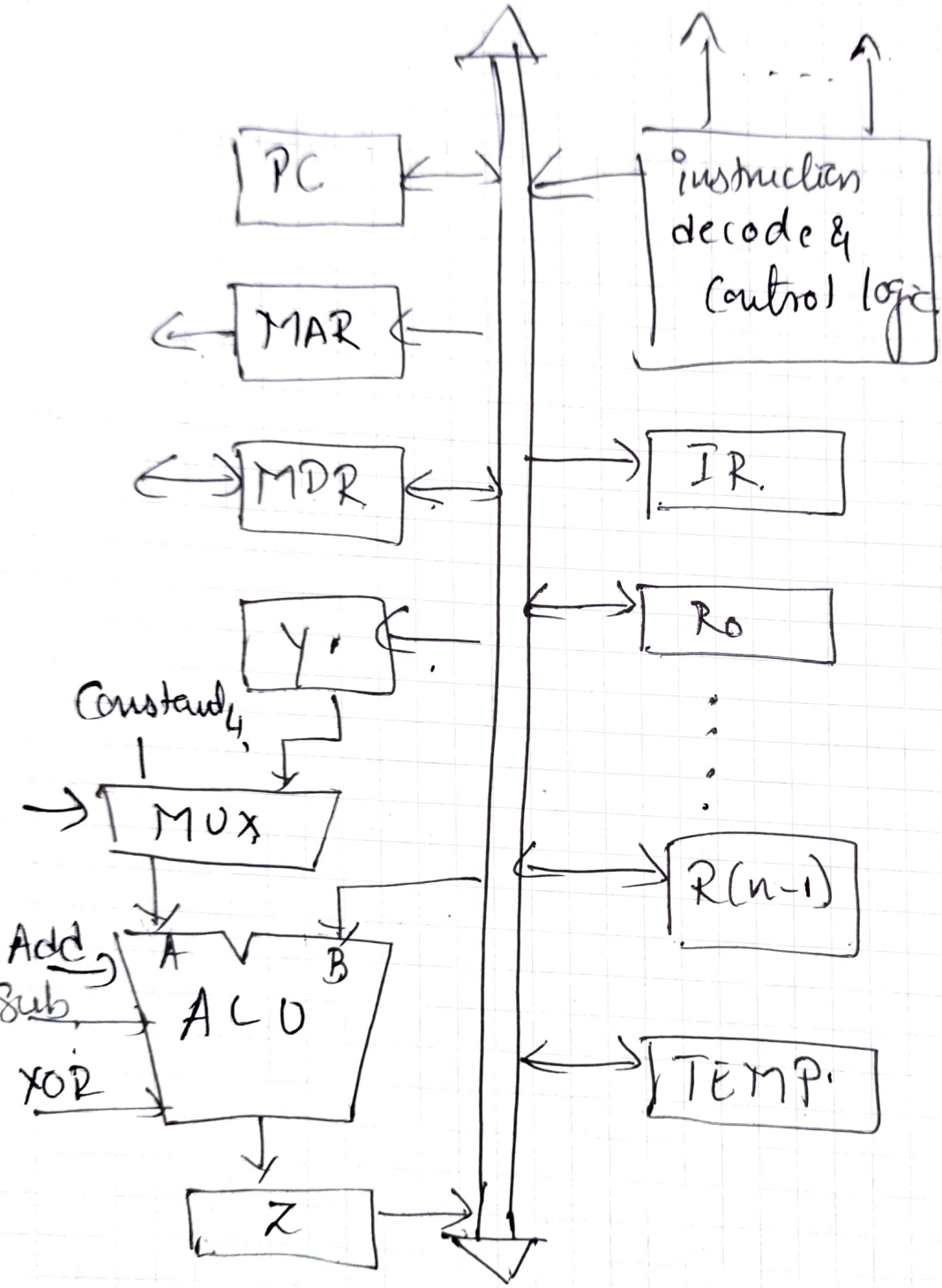
Memory Address	Control field	Address field
	C_0 C_1 C_2 C_3 C_4 C_5 C_6 C_7 C_8 C_9 C_{10}	
Eg 1. 0000	0 0 1 1 0 0 1 0 1 1 0	0 0 0 1
2. 0001	1 0 0 1 1 1 0 0 1 0 1	0 0 1 0
3. 0 0 1 0	1 1 0 0 0 1 1 0 0 1 0 1	0 0 1 1

Micro Instruction consists of

1. one or more micro-operations to be executed
2. Address of next micro-instruction

Add R1, M

	or Pcout, MARin, Read, Select4, Add, Zin.
1. PC → MAR	Pcout, MARin, Read. Clear y, set Cin, Add Zin.
2. M → MDR PC ← PC+1	Zout, PCin, WMFC or yin
3. MAR → IR	MARout IRin.
4. R1 → X	R1out, Xin., CLRC
5. M → ALU	Mout, ADD, Zin.
6. Z → R1	Zout, R1in


PC
MAR
MDR
Y
Constant 4
MUX
Add
Sub
XOR
A
B
ALU
Z
instruction decode & Control logic
IR
R0
R(n-1)
TEMP

Add $(R_3), R_1$

Pcout, MARin, Read, Select 4, Add, Zin.

Zout, Pcin, Yin, WMFC

MDRout, IRin.

R3out, MARin, Read.

R1out, Yin WMFC.

MDRout, Select Y, Add, Zin

Zout, Rin, End.

MUL R_1, R_2 $\boxed{R_1 \leftarrow R_1 * R_2.}$

1. Pcout, MARin, Read, clear Y, set Cin, Add Zin
2. Zout, PCin, Wait for memory fetch cycle
3. MDRout IRin
4. ~~RR~~ R1out, Xin, CIRC
5. R2 out, MUL, Zin

Chapter 5

Memory Organization

CE – SE – Digital Logic & Computer Architecture

Prof. Rasika Malgi

Dept. of AI &DS

SIES Graduate School of Technology

Memory Technology

- Computer needs to store the instructions and data required for its operation
- This instruction and data are stored in memory
- **memory** refers to the computer hardware integrated circuits that store information for immediate use in a computer**.**

Memory types

1. CPU registers:

- High speed registers for temporary storage of instruction and data
- They form a general purpose register file for a storing data as it is processed
- Capacity of register file is 32 words
- Each register can be accessed, i.e read or write within a single clock cycle

2. Main (Primary) Memory

- It is large and fast external memory stores data and programs that are active in use
- Storage locations in MM are addressed directly by the CPU load and store instruction
- Capacity of MM: between 1 and 2^{10} megabytes
- Megabyte 1 MB is 2^{20} bytes
- And 2^{10} MB=2^{30} bytes(referred as 1GB)
- Access time of 5 or more clock cycles

3. Secondary Memory

- This memory type is larger in capacity but slower than MM
- It stores a system programs, large data files that is not continuously required by CPU
- Representative Technologies for secondary memory are magnetic hard disks and CDROMs
- Capacity: many gigabytes
- Access time: Measured in miliseconds

4. Cache

- Most computers have another level of IC memory----sometimes several such levels called cache memory
- Which is logically placed between CPU registers and MM
- Cache capacity is less than the MM
- Access time of 1 to 3 cycles
- Cache is much faster than MM
- Caches are essential components of high-performance computer that aims to make CPI<=1 (cycles per instruction)

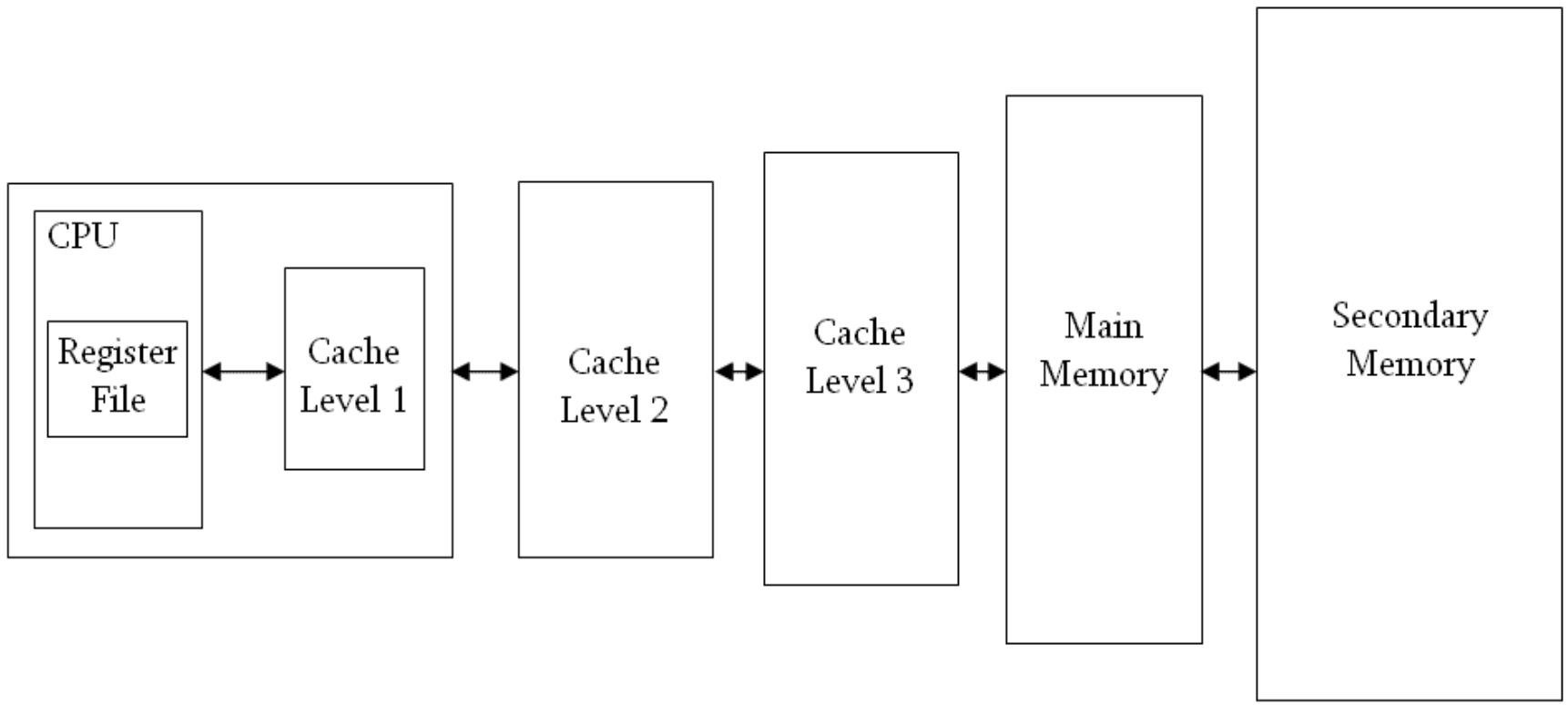


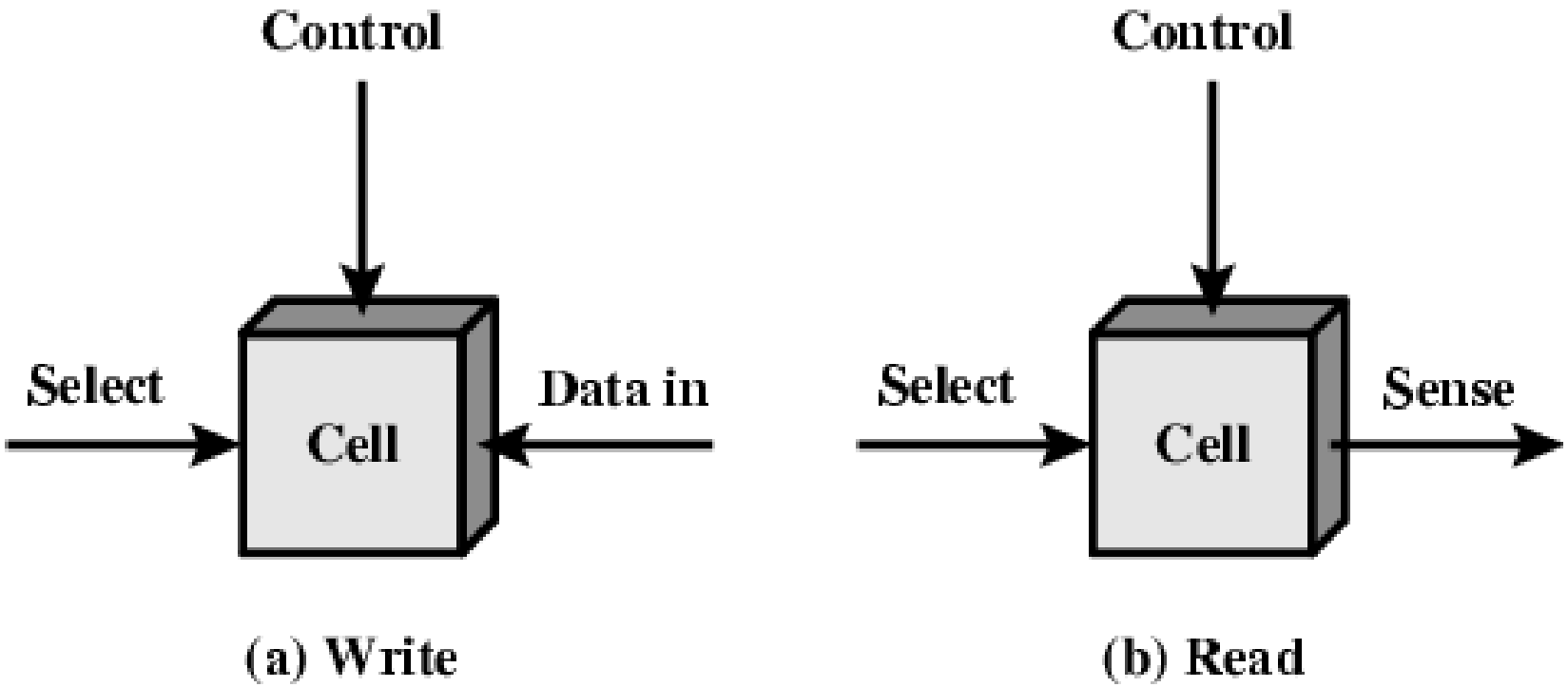
Figure : Conceptual Organization of Multilevel Memories in a Computer System

Semiconductor Main Memory

- Basic element of a Semiconductor memory is the memory cell
- All semiconductor memory cells share a **properties**:

1. They exhibit two stable(or semistable) states, which can be used to represent binary 1 and 0.
2. They are capable of being written into(at least once) the state
3. They are capable of being read to sense the state**.**

Memory Cell Operation



Contd..

- Cell has 3 functional terminals carrying an electrical signal

1. Select terminal: Selects a memory cell for a read or write operation

2. Control terminal: indicates read or write

For writing: sets the state to 1 or 0

For reading: used for output of the cell’s state

Memory Parameters

1) Capacity

- L X W bits
- L= Number of locations (address)
- W= width of each location

- e.g. l=8, w= 8 bits
- Therefore, Capacity= 8 * 8=64 bits

- **2) Speed**

- Two parameters are:
- - Access time tA
- - Cycle time tC

- **Cycle Time**
- To perform read operation, first the address of the location is given to memory followed by read control signal. Time taken by memory to complete read operation after receiving read control signal
-

- **3)Latency**
- When we want to access data from hard disk,first access has larger access time whereas successive location have shorter access time.
- Latency- time required for first access time
- E.g. Hard disk

- **4)Bandwidth**
- Data transfer rate by memory , expressed as no. of bytes per second.

Key characteristics of memory

- **1) Location:**
- Internal-processor,reg ,mm,cache memory
- External- optical, Magnetic tapes

- 2**) Capacity**
- Number of words, bytes
- Internal- Bytes
- External- Number of words

- **3) Unit of transfer**
- Word,Block

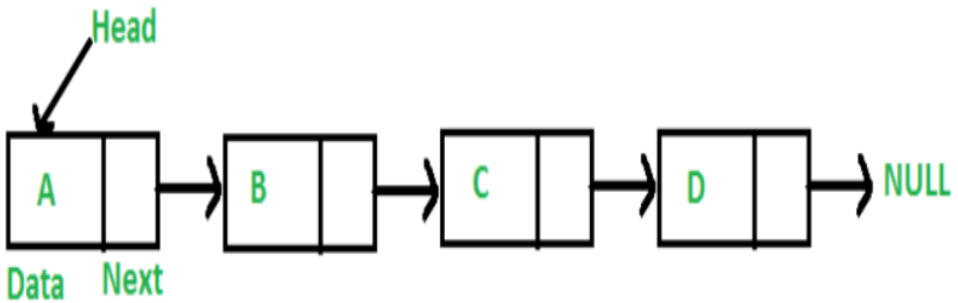
- The unit of transfer is the number of bits read out of or written into memory at a time.
- For internal memory the unit of transfer is equal to number of electrical lines into and out of the memory module (e.g.word)
- For external memory unit of transfer is blocks.

- **4)Access Method**

- a)Sequential – e.g. Tape
- b)Random access
- c) Direct Access(combination of random access and sequential access)
- d)Associative Access
- Key – tag word
- tag word
- |
- Match

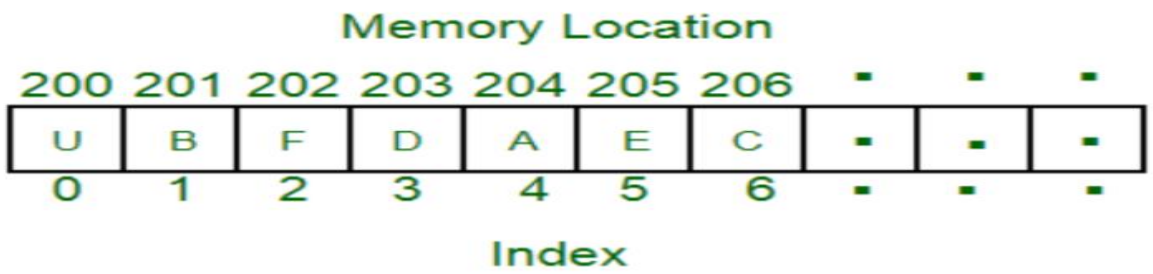
- **5) Performance**
- Access time, Cycle time, transfer rates
- **6) Physical Type(Semiconductor, magnetic, Optical)**
- **7) Physical Characteristics** (Volatile, Non volatile)
- **8) Organisation** (Erasable/Non erasable)

a) Sequential Access (Like singly linked list)



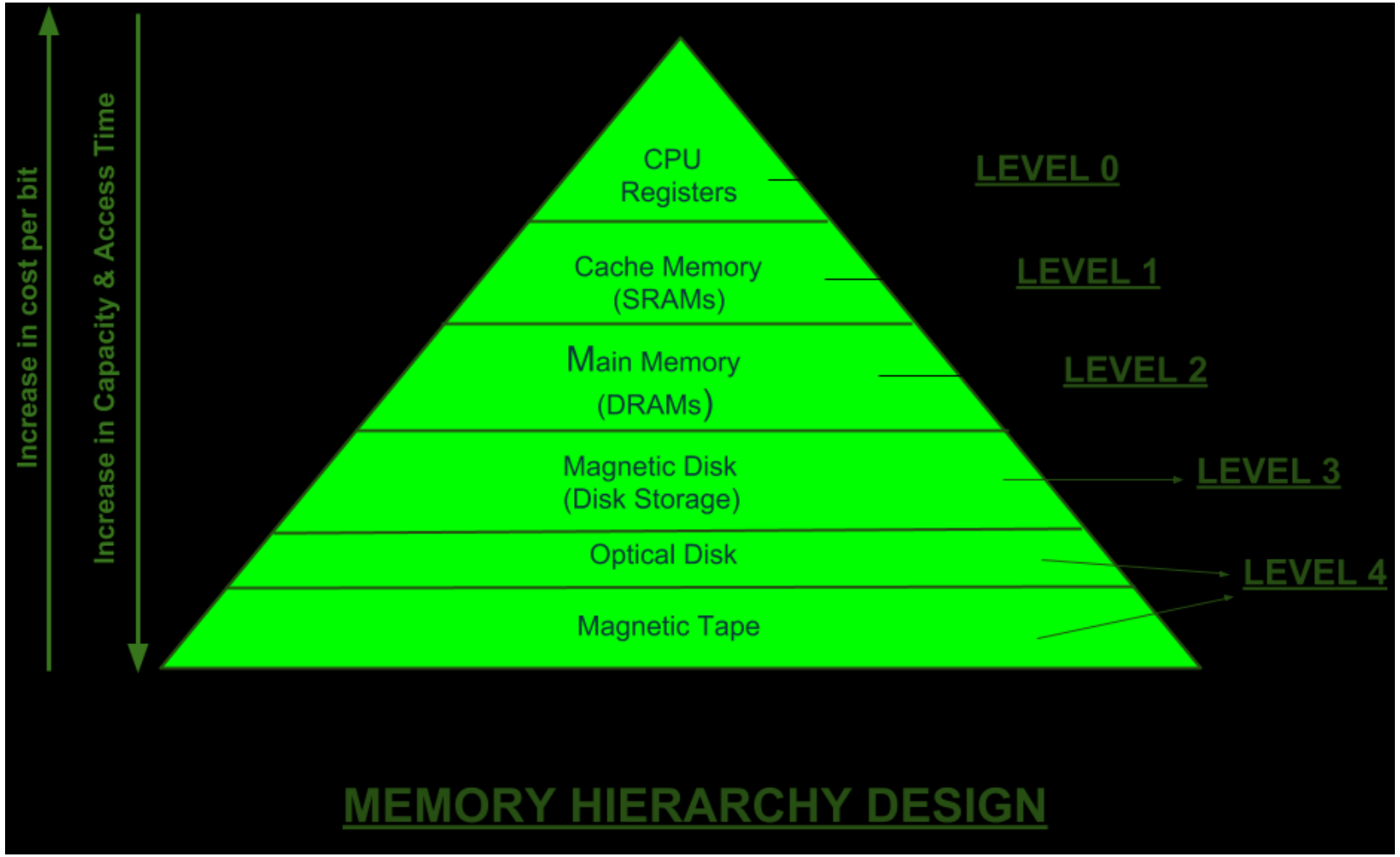
- Applications of this sequential memory access are magnetic tapes, magnetic disk and optical memories.

- b) Random Access (Like Array)



- Applications of this random memory access are RAM and ROM. Independent of physical location.

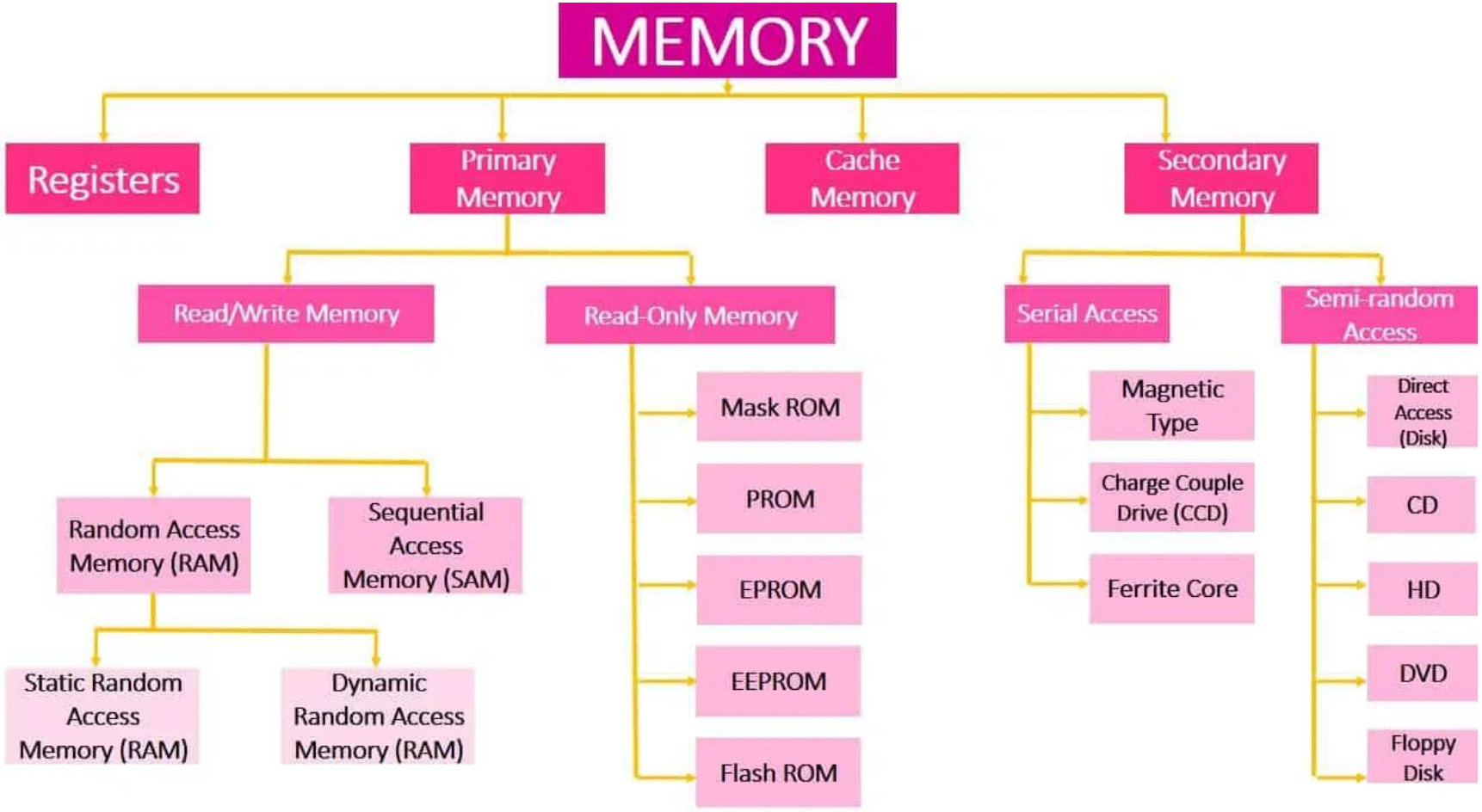
Memory Hierarchy



- To achieve the performance the memory hierarchy in above figure should be followed which results in

1) Decreasing cost per bit
2) Increasing capacity
3) Increasing access time
4) Decreasing frequency of access of the memory by the processor

Classification of memory



Semiconductor Memory

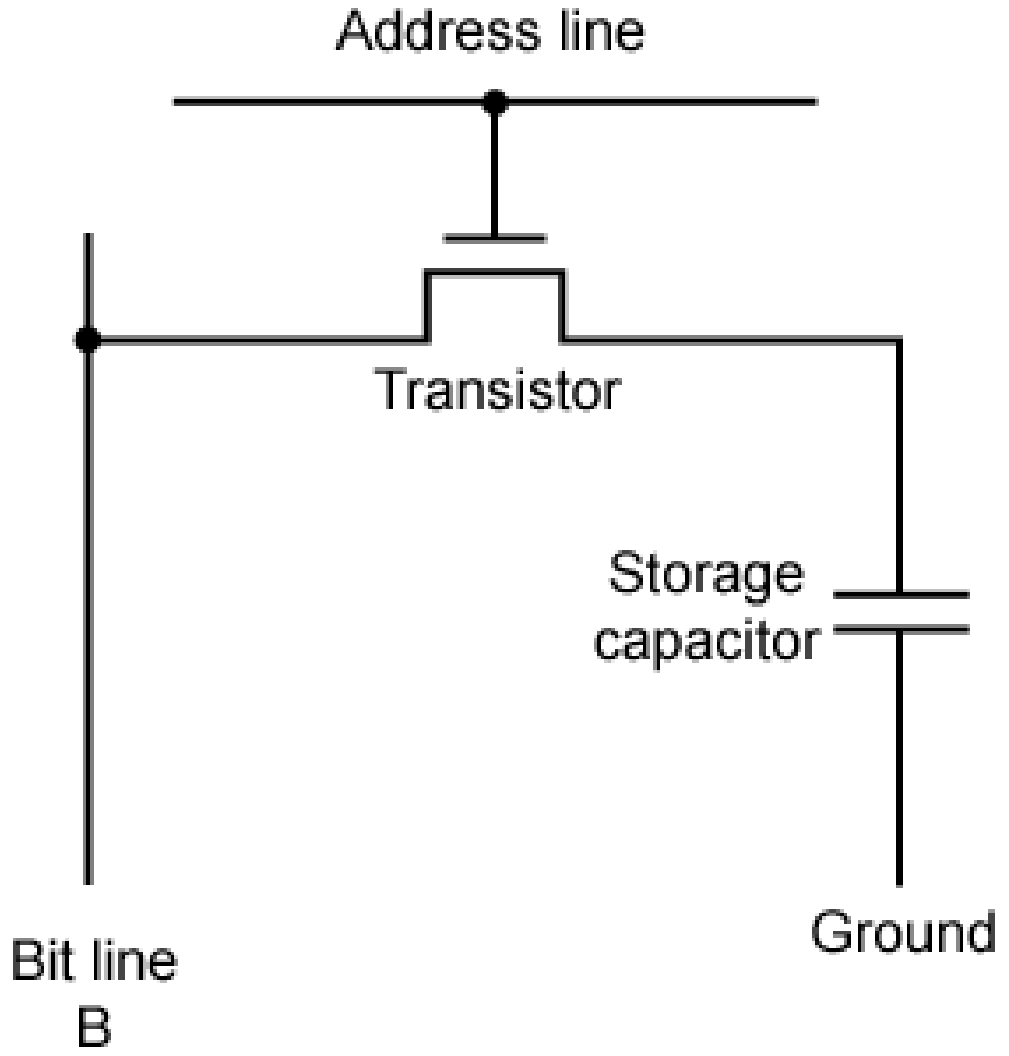
❖RAM

- All semiconductor memory is random access
- Read/Write
- Volatile
- Temporary storage
- Static or dynamic (SRAM or DRAM)

Dynamic RAM

- Bits stored as charge in capacitors
- Charges leak
- Need periodic refreshing to maintain data storage
- Simpler construction
- Smaller per bit
- Less expensive
- Need refresh circuits
- Slower
- Main memory

Dynamic RAM Structure



DRAM Operation

- Address line active when bit read or written
  - Transistor switch closed (current flows)
- Write
  - Voltage to bit line
    - High for 1 low for 0
  - Then signal address line
    - Transfers charge to capacitor
- Read
  - Address line selected
    - transistor turns on
  - Charge from capacitor fed via bit line to sense amplifier
    - Compares with reference value to determine 0 or 1
  - Capacitor charge must be restored

Static RAM

- Bits stored as on/off switches
- No charges to leak
- No refreshing needed when powered
- More complex construction
- Larger per bit
- More expensive
- Does not need refresh circuits
- Faster
- Cache

Sr. no	SRAM	DRAM
1	No refreshing required	Continuous refreshing is required
2	It is faster for accessing data	It is slower in accessing data
3	It takes more space on chips as more number of components are required per bit	It takes less space on chip as less number of components are required per bit
4	costly	cheaper
5	Bit density is lesser	Bit density is more
6	The bit is stored in a flipflop	Bit is stored as a charged or discharged capacitor
7	SRAM used for cache memory	DRAM is used for semiconductor memory

Read Only Memory (ROM)

- Contains permanent pattern of data that cannot be changed
- Permanent storage
  - Nonvolatile(no power source is required to maintain the bit values in memory)
- Application of ROM is
  1. Microprogramming
  2. Library subroutines
  3. Systems programs (BIOS)
  4. Function tables

PROM(Programmable ROM)

- When a small no. of ROMs with a particular memory content is needed
- Is less expensive
- Non volatile
- Written only once
- The writing process is performed electrically

Erasable Programmable ROM

- Is read and written electrically
- Before a write operation , all the storage cells must be erased to the same initial state by exposure of package chip to UV radiation
- EPROM can be altered multiple times

Electrically Erasable programmable ROM

- Read mostly memory that can be written into at anytime without erasing prior contents
- Only bytes by bytes are updated
- Write operation takes longer time than read operation

Advantage

1. Nonvolatility with flexibility of being updatable in place

- But More expensive

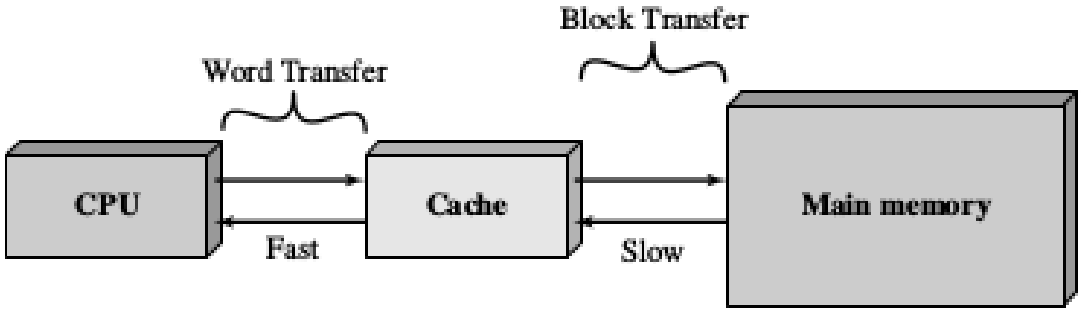
Flash Memory

- Semiconductor memory is flash memory
- Uses electrical erasing technology
- An entire memory is erased in 1 or few seconds

Which is faster than EPROM

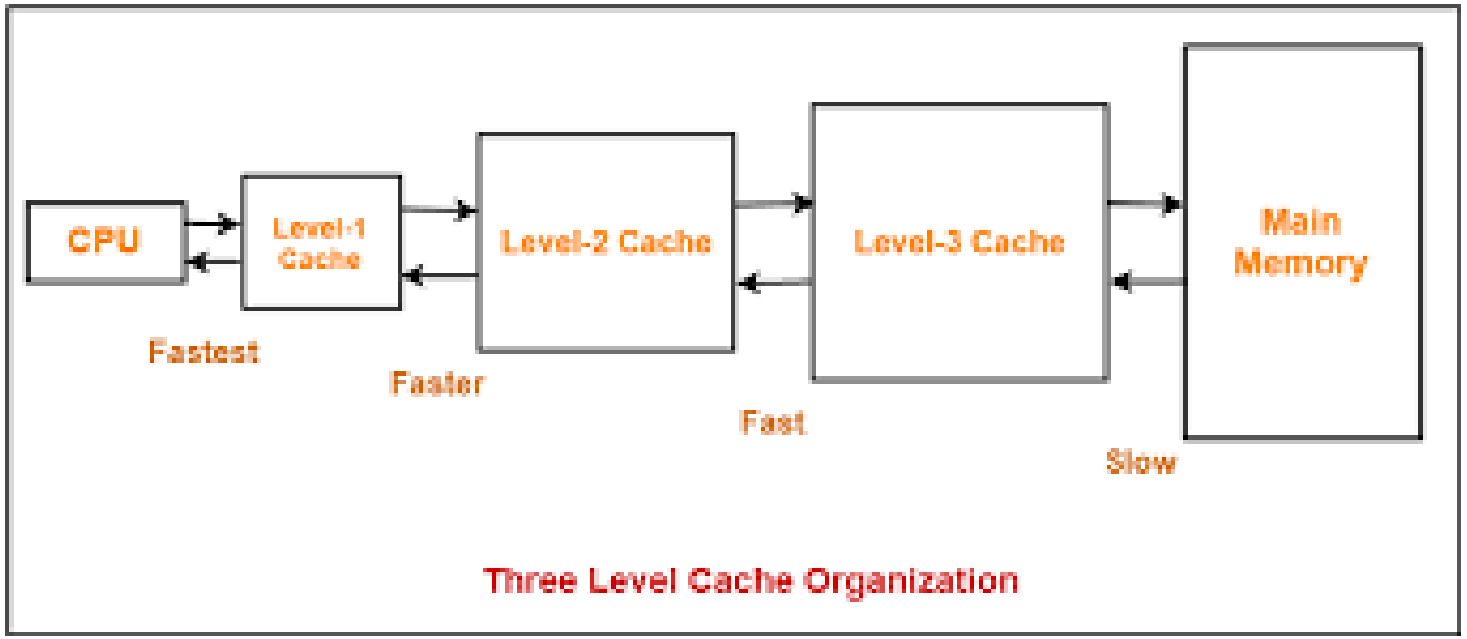
It is possible to erase just blocks of memory rather than an entire chip.

Cache memory principles



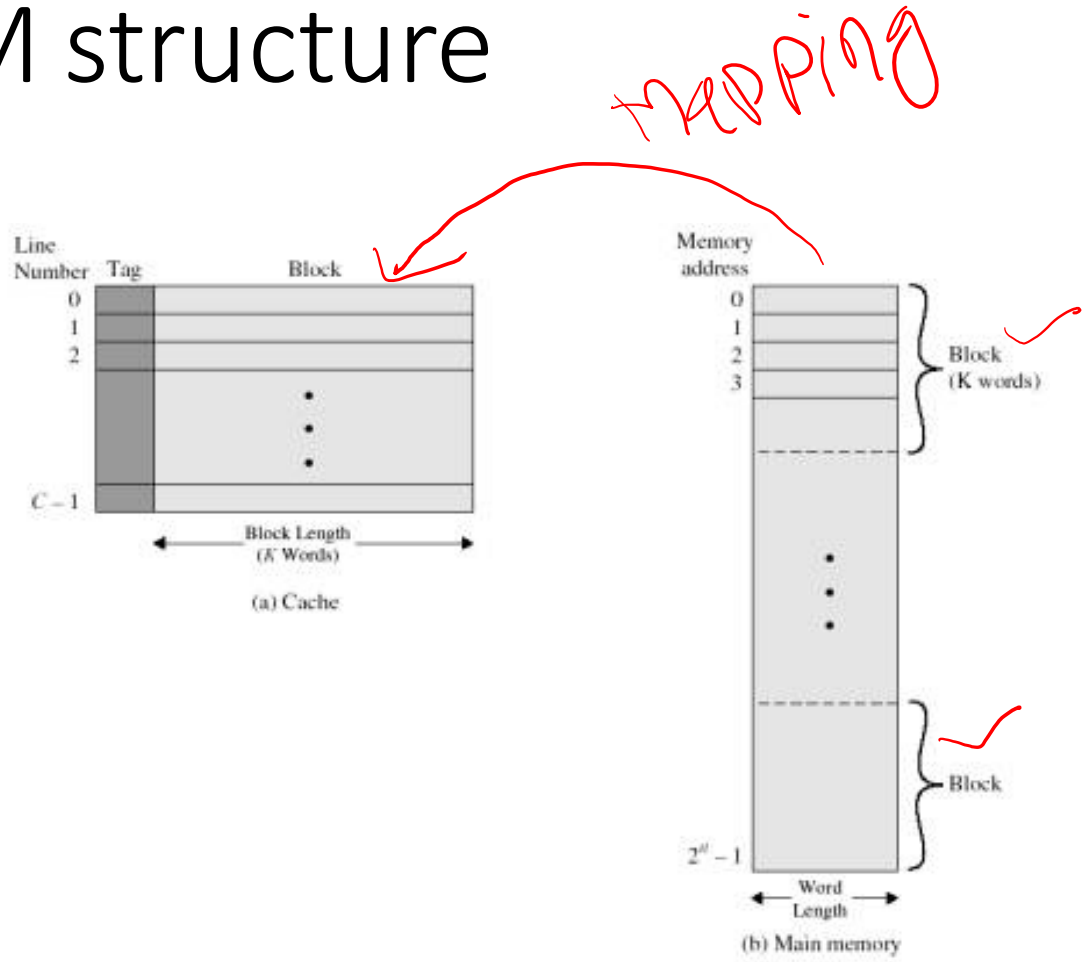
(a) Single cache

- Goal: Memory speed
- Cache contains copy of portion of MM
- When processor wants to read data then first check is made into cache memory and then MM



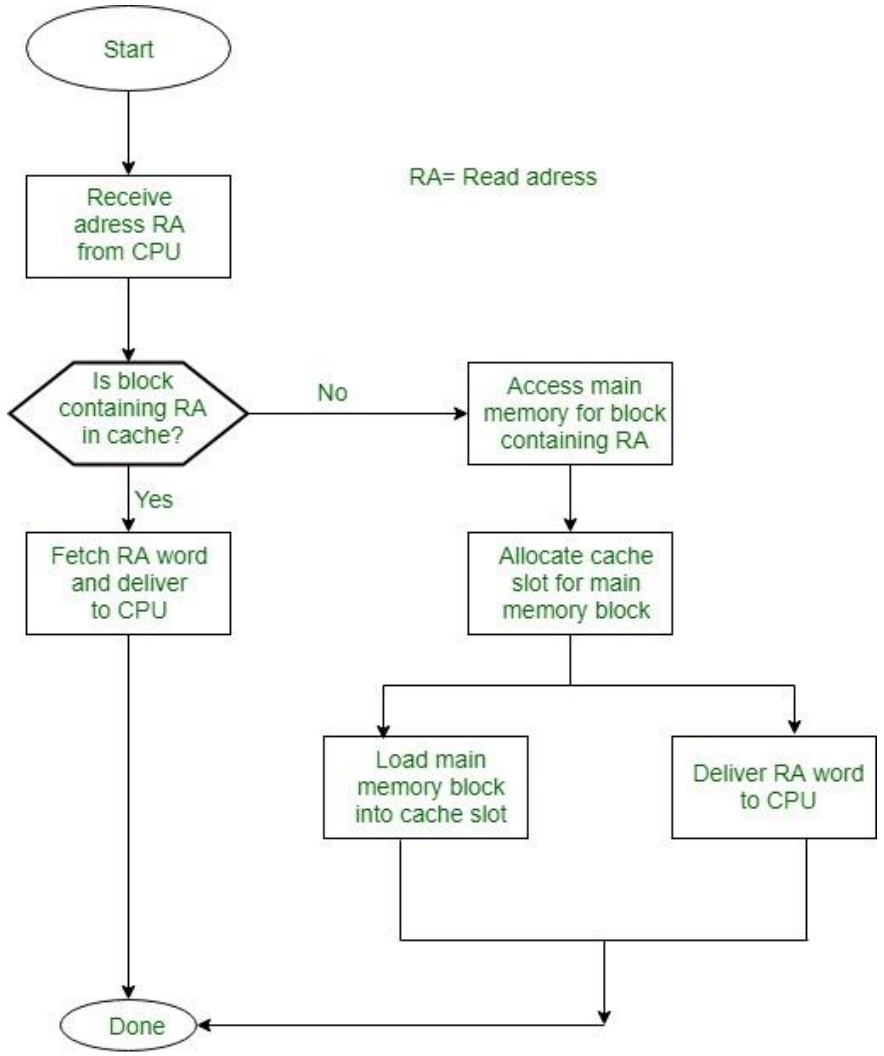
Three Level Cache Organization

Cache/MM structure



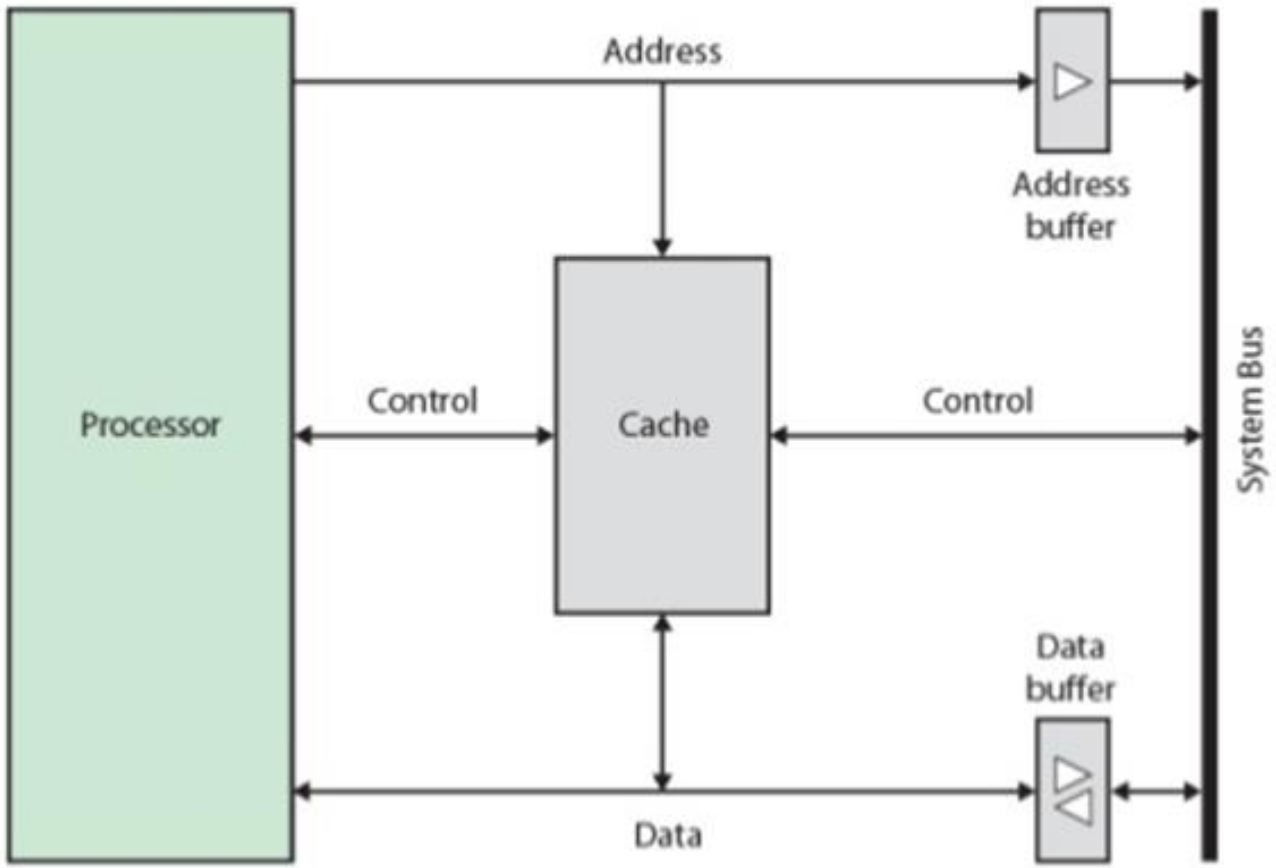
- MM consist of upto 2^n addressable words, with each word having unique n bit address
- MM has fixed length block of K words each.
- M=2^n/K
- No. of MM blocks = Total number of words/ No. of words in each block
- Cache consist of m blocks, called lines
- Each line has K words plus a tag of few bits
- Each line has control bits to indicate whether line has modified since being loaded into cache
- Length of line not including tag and control bits called line size
- Number of lines < No. of MM blocks

Cache Read Operation



Cache Read Operation

TYPICAL CACHE ORGANIZATION



Elements of Cache Design

1) Cache size

- Cache size should be:

❖ Small enough so that overall:

- • Average cost per bit is close to that of main memory alone;

❖ Large enough so that the overall

- • Average access time is close to that of the cache alone;

- The larger the cache, the more complex the addressing logic:

- • Result: large caches tend to be slightly slower than small ones

- Available chip and board area also limits cache size.

- 2) Cache addresses
- There are two types of cache addresses:
- • Physical addresses:
- Actual memory addresses;
- • Logical addresses:
- •Virtual-memory addresses;

✓ Virtual memory performs mapping between:

- **Logical addresses** used by a program into **physical addresses**.

- **Main idea behind virtual memory:**
- • Disregard amount of main memory available;
- •Transparent transfers to/from:

✓ main memory and

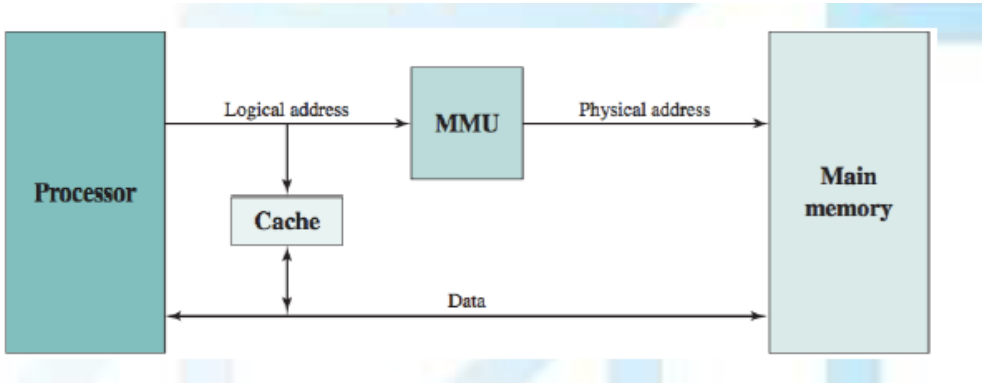
✓ secondary memory

✓ **Idea:** use RAM, when space runs out use HD

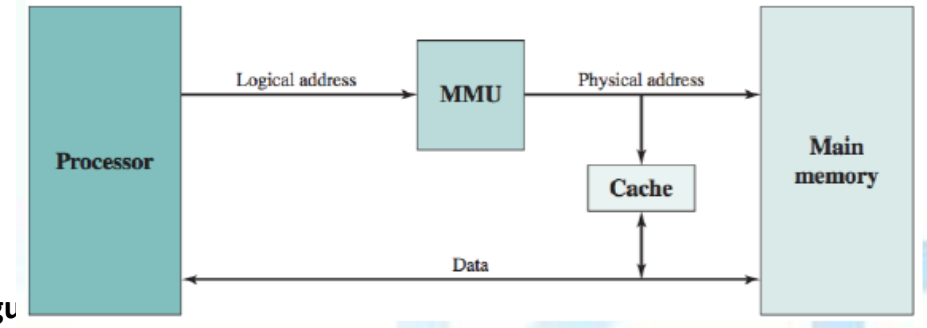
- Requires a hardware memory management unit (MMU):

✓ To translate virtual addresses into a physical addresses;

- With virtual memory cache may be placed:
- between the processor and the MMU;



- **Figure: Virtual Cache (Source: [Stallings, 2015])**
- between the MMU and main memory;



- **Figu**

- **Virtual cache** stores data using logical addresses.
- • Processor accesses the cache directly, without going through the MMU.
- • **Advantage:**
- • Faster access speed;
- • Cache can respond without the need for an MMU address translation;
- • **Disadvantage:**
- • Same virtual address in two different applications refers to two different
- physical addresses;
- • Therefore cache must be flushed with each application context switch or extra bits must be added to each cache line to identify which virtual address space this address refers to.

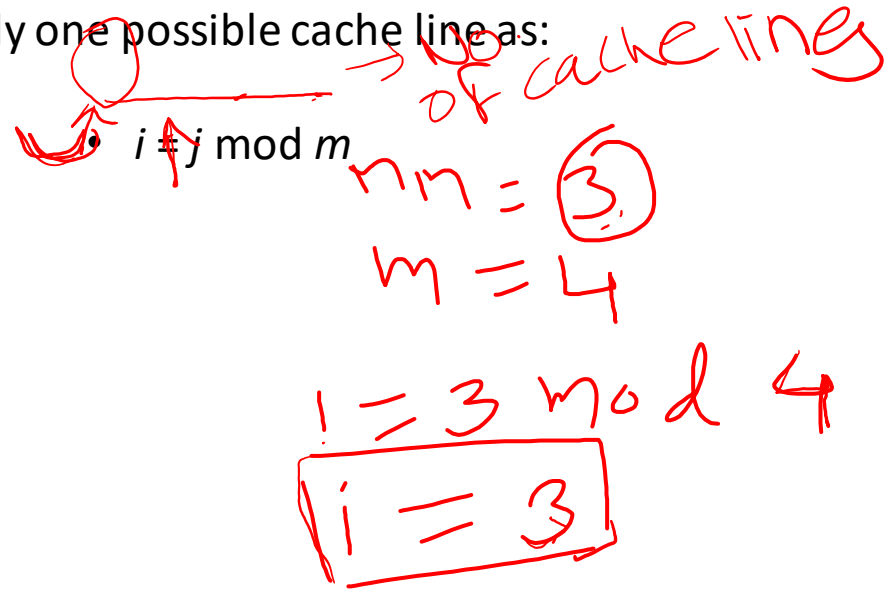
3) Mapping

- Recall that there are fewer cache lines than main memory blocks
- How should one map main memory blocks into cache lines? Any ideas?

- Three techniques can be used for mapping blocks into cache lines:
- • Direct;
- • Associative;
- • Set associative

a) Direct Mapping

- Maps each block of main memory into only one possible cache line as:
  - $i = j \bmod m$



- where:
  - i = cache line number;
  - j = main memory block number;
  - m = number of lines in the cache

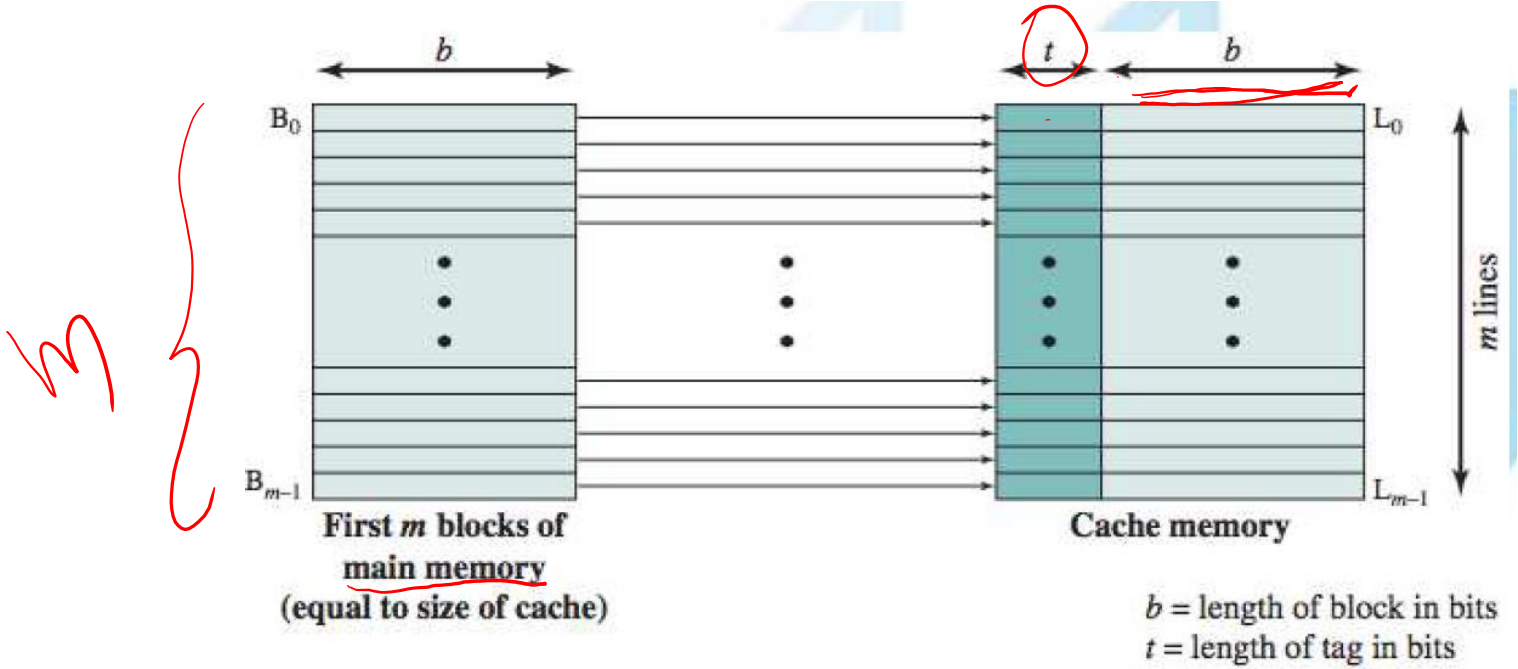


Figure: Direct mapping (Source: (Stallings, 2015))

- Previous picture shows mapping of main memory blocks into cache:
- First *m* main memory blocks map into each line of the cache;
- Next *m* blocks of main memory map in the following manner:
- *Bm* maps into line *L*0 of cache;
- *Bm*+1 maps into line *L*1;
- and so on...
- Modulo operation implies repetitive structure;

- With direct mapping blocks are assigned to lines as follows:

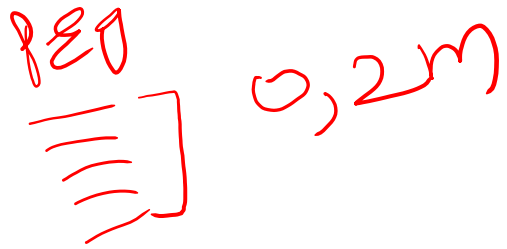
Cache line	Main memory blocks assigned
0	$0, m, 2m, \ldots, 2^s - m$
1	$1, m + 1, 2m + 1, \ldots, 2^s - m + 1$
⋮	⋮
$m - 1$	$m - 1, 2m - 1, 3m - 1, \ldots, 2^s - 1$

$i = j \bmod m$

Over time:

- Each line can have a different main memory block;
- We need the ability to distinguish between these;
- Most significant bits, the **tag**, serve this purpose.

- Direct mapping technique:
- **Advantage:** simple and inexpensive to implement;
- **Disadvantage:** there is a fixed cache location for any given block;
- if a program happens to reference words repeatedly from two different
- blocks that map into the same line;
- then the blocks will be continually swapped in the cache;
- hit ratio will be low (a.k.a. **thrashing**).



b) Associative Mapping

- Overcomes the disadvantage of direct mapping by:
- permitting each block to be loaded into any cache line:

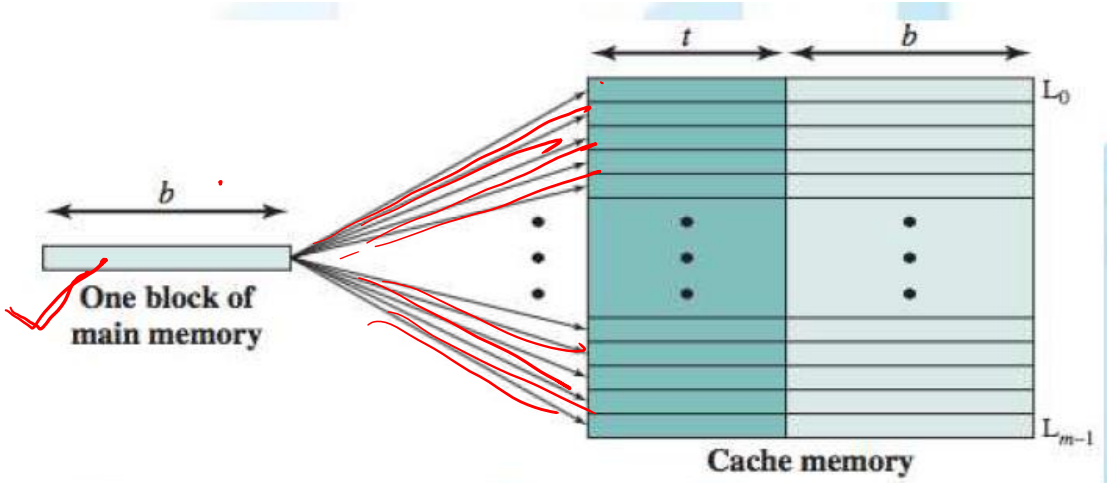
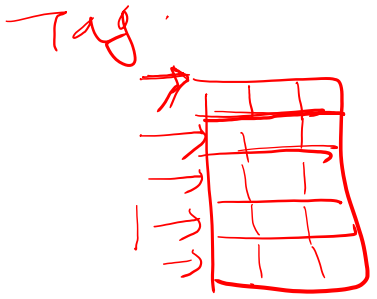


Figure: Associative Mapping (Source: (Stallings, 2015))

- Cache interprets a memory address as a **Tag** and a **Word** field:
- **Tag:** (*s* bits) uniquely identifies a block of main memory;
- **Word:** (*w* bits) uniquely identifies a word within a block;



- To determine whether a block is in the cache:
- simultaneously **compare** every line's tag for a match:
- If a **match exists**, then **Cache Hit:**

  Use the tag field of the memory address to index the cache line;

  Retrieve the corresponding word from the cache line;

- If a **match does not exist**, then **Cache Miss:**

  Choose a cache line. How?

  Update the cache line (word + tag);

- **Advantage:**
- Flexibility as to which block to replace when a new block is read into the
- cache;

- **Disadvantage:**
- Complex circuitry required to examine the tags of **all** cache lines in
- parallel.

c) Set Associative Mapping

- Combination of direct and associative approaches:
- Cache consists of a number of sets, each consisting of a number of lines.

❖ From **direct mapping:**

- each block can only be mapped into a single set;
- *I.e.* Block *Bj* always maps to set *j*;
- Done in a modulo way =)

❖ From **associative mapping:**

- each block can be mapped into any cache line of a certain set.

- The relationships are:
- $m = v \times k$
- $i = j \bmod v$
- where:
- i = cache set number;
- j = main memory block number;
- m = number of lines in the cache;
- v = number of sets;
- k = number of lines in each set

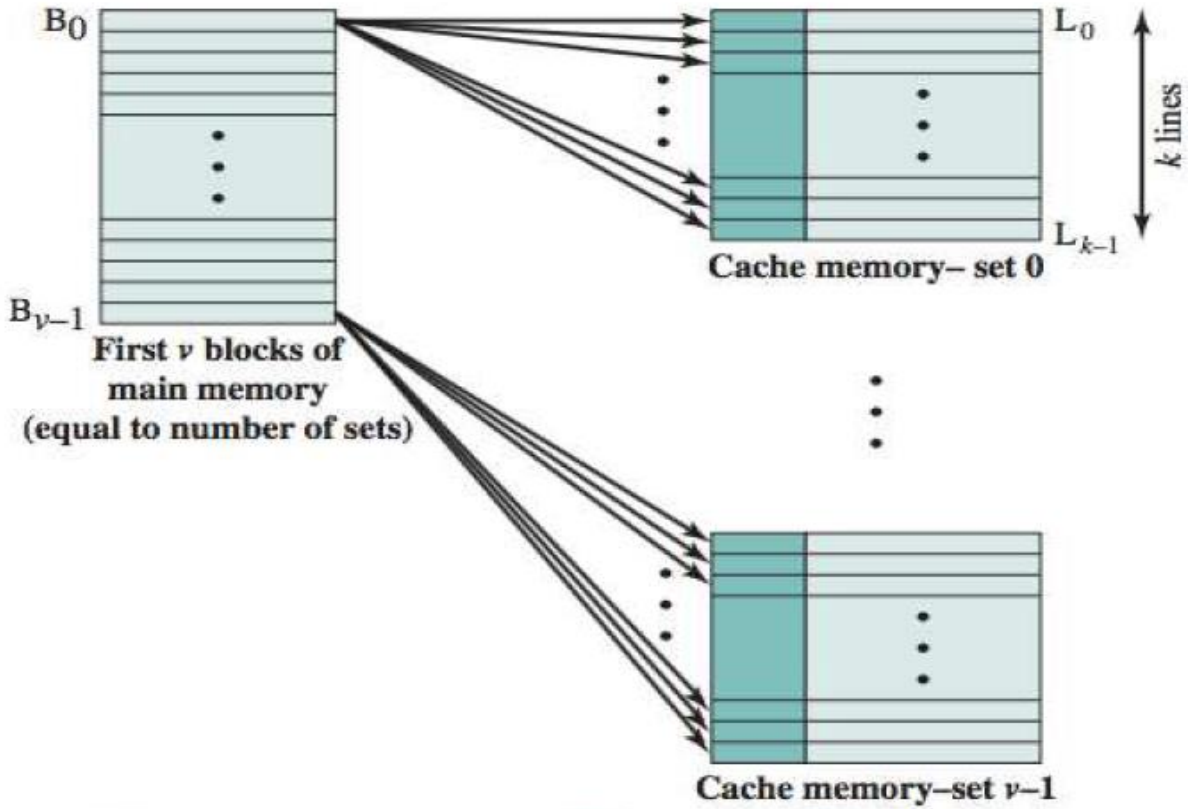


Figure: *v* associative mapped caches (Source: (Stallings, 2015))

- Cache interprets a memory address as a **Tag**, a **Set** and a **Word** field:
- **Set:** identifies a set (d bits, $v = 2d$ sets);
- **Tag:** used in conjunction with the set bits to identify a block ($s - d$ bits);
- **Word:** identifies a word within a block;

- To determine whether a block is in the cache:

  Determine the **set** through the set fields;

  Compare address tag simultaneously with all cache line tags;

  If a **match exists**, then **Cache Hit:**

  Retrieve the corresponding word from the cache line;

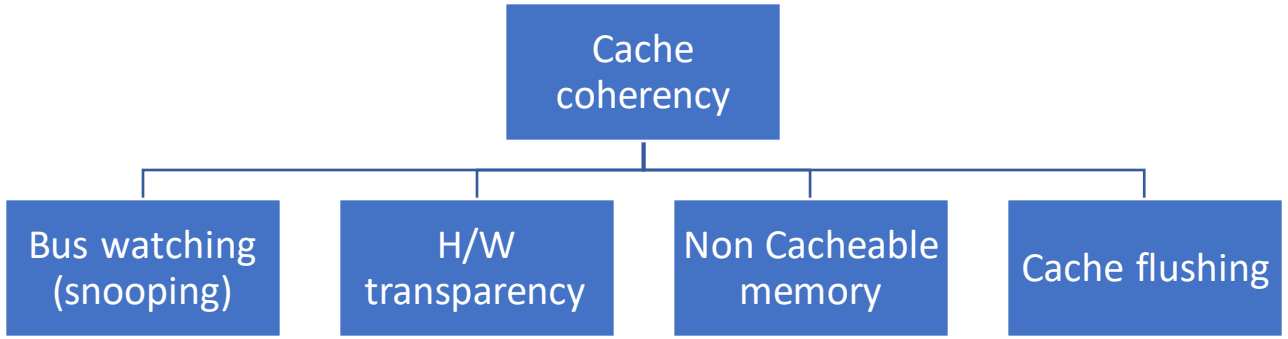
  If a **match does not exist**, then **Cache Miss:**

  Choose a cache line within the set. How?

  Update the cache line (word + tag);

Cache Coherency

- In a single CPU system two copies of same data, one in cache memory and another in MM may become different. This data inconsistency is called as **cache coherence problem.**
- There are four different approaches to prevent data inconsistency, i.e, to protect cache coherency



Cache updating policies

- In cache system, two copies of same data can exist at a time, one in cache and one in MM . If one copy is altered and other is not, two different sets of data become associated with the same address.
- To prevent this, the cache system has updating systems such as :

1) write through system
2) Buffered write through system
3) Write – back system

1) Write – through system

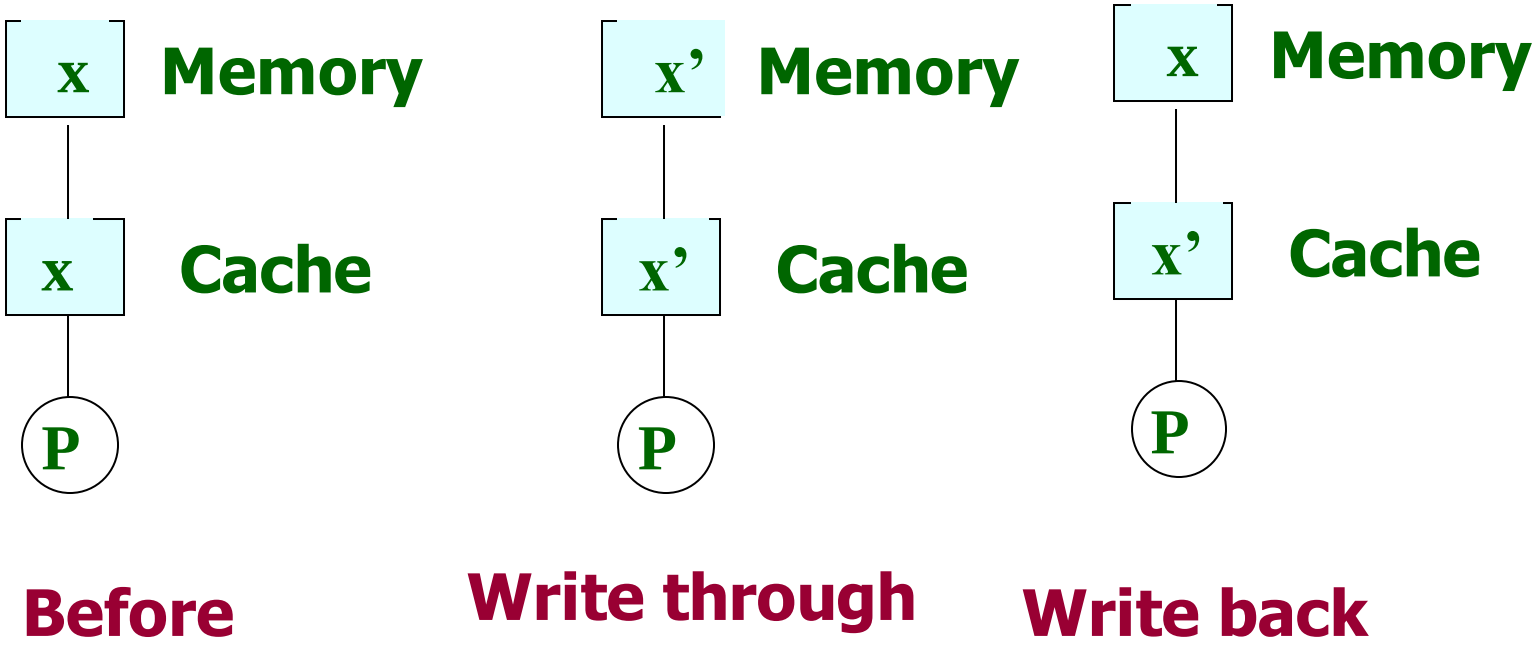
- The cache controller copies data to MM immediately after it is written to the cache
- Due to this, MM always contains a valid data and thus any block in cache can be overwritten immediately without data loss
- Simple approach
- This approach requires time to write data in MM with increase in bus traffic
- It reduces system performance

- **2) Buffered write through system**
- In buffered write through system, the processor can start a new cycle before the write cycle to MM is completed
- Means write accesses to MM are buffered
- In such systems, read access which is a “cache hit” can be performed simultaneously when MM is updated
- Two consecutive write operations to MM or read operation with cache “miss” require the processor to wait

- **3) Write back system**
- In this the alter(control/update) bit in the tag field is used to keep information of the new data
- If it is set, the controller copies the block to MM before loading new data into cache
- Due to one time write operation, number of write cycles are reduced in write back system
- **Disadvantage:**

1. Complex logic
2. It is necessary that, all altered blocks must be written to the MM before another device can access these blocks in MM
3. In case of power failure data in cache memory is lost, so there is no way to tell which locations of the MM contain old data. Therefore, the MM as well as cache must be considered volatile and provisions must be made to save the data in the cache

Writing in the cache

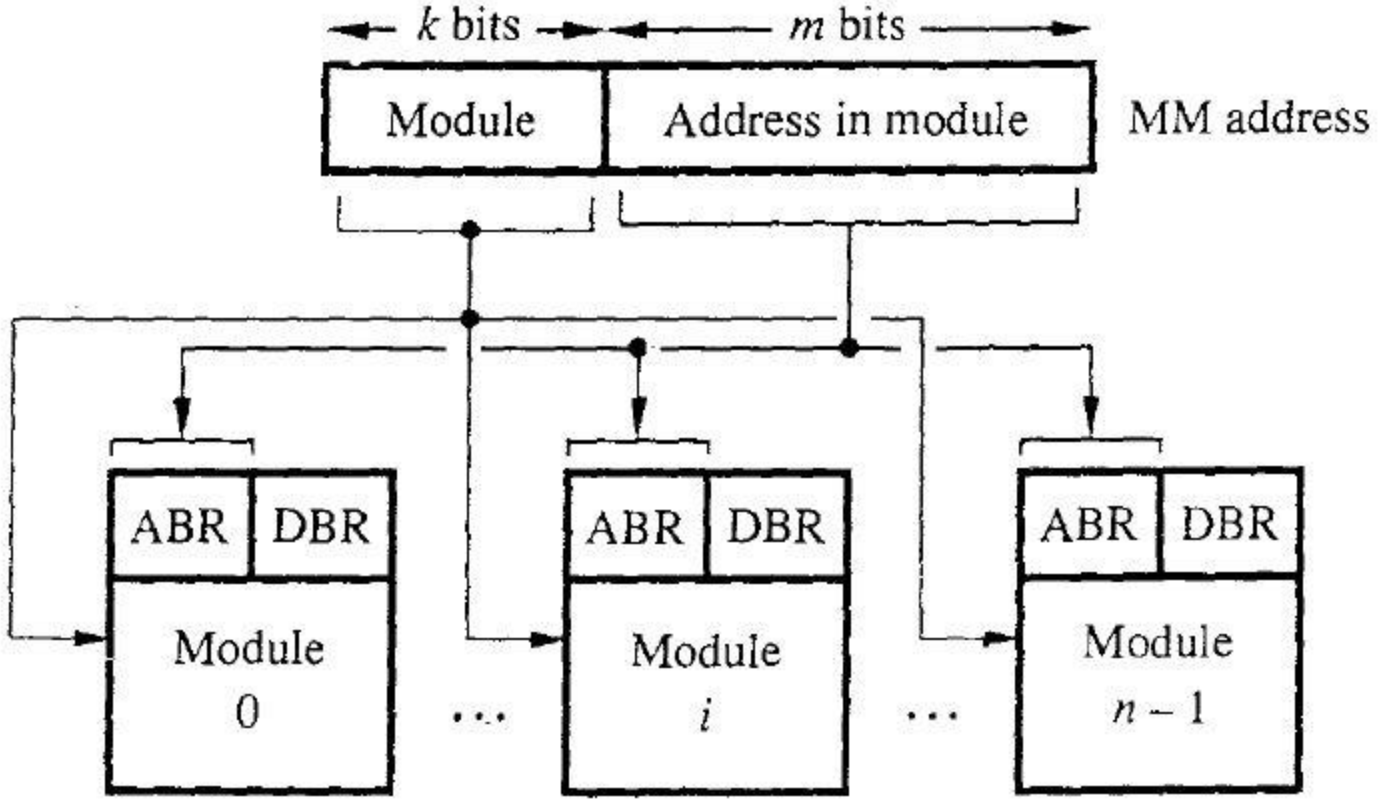


Memory Interleaving and Associative Memory

Memory Interleaving

- A single memory module causes sequential access (only one memory access performed at a time) => not efficient.
- Memory interleaving
  - Splits memory across multiple memory modules (or banks)
  - Can increase efficiency => Can issue memory requests to all banks/modules at the same time.
- Access is more efficient when memory is organized into banks of chips with the addresses interleaved across the chips
  - Low-order interleaving (LOI) =>the low order bits of the address are used to select the memory bank.
  - High-order interleaving (HOI)=> the high order bits of the address are used to select the memory bank.

interleaved memory



(a) Consecutive words in a module

Memory Interleaving

- Organize memory chips in modules/banks and issue memory requests to all banks at the same time.
- Hence if you buy memory to upgrade you buy a Memory Module/Bank.
- Access is more efficient when memory is organized into banks of chips with the addresses interleaved across the chips
- Low-order interleaving, the low order bits of the address specify which memory bank contains the address of interest.
- High-order interleaving, the high order address bits specify the memory bank/module.

Example 1

- Say we have a byte-addressable memory consisting of 8 modules/banks of 4 bytes each.
  - This gives a total of = 4 * 8 = 32 bytes of memory.
  - To identify each byte , we need 5 bits ($2^5 = 32$)

3 bits are used to determine the module there are 8 modules , so 2^3

2 bits to determine offset within the module

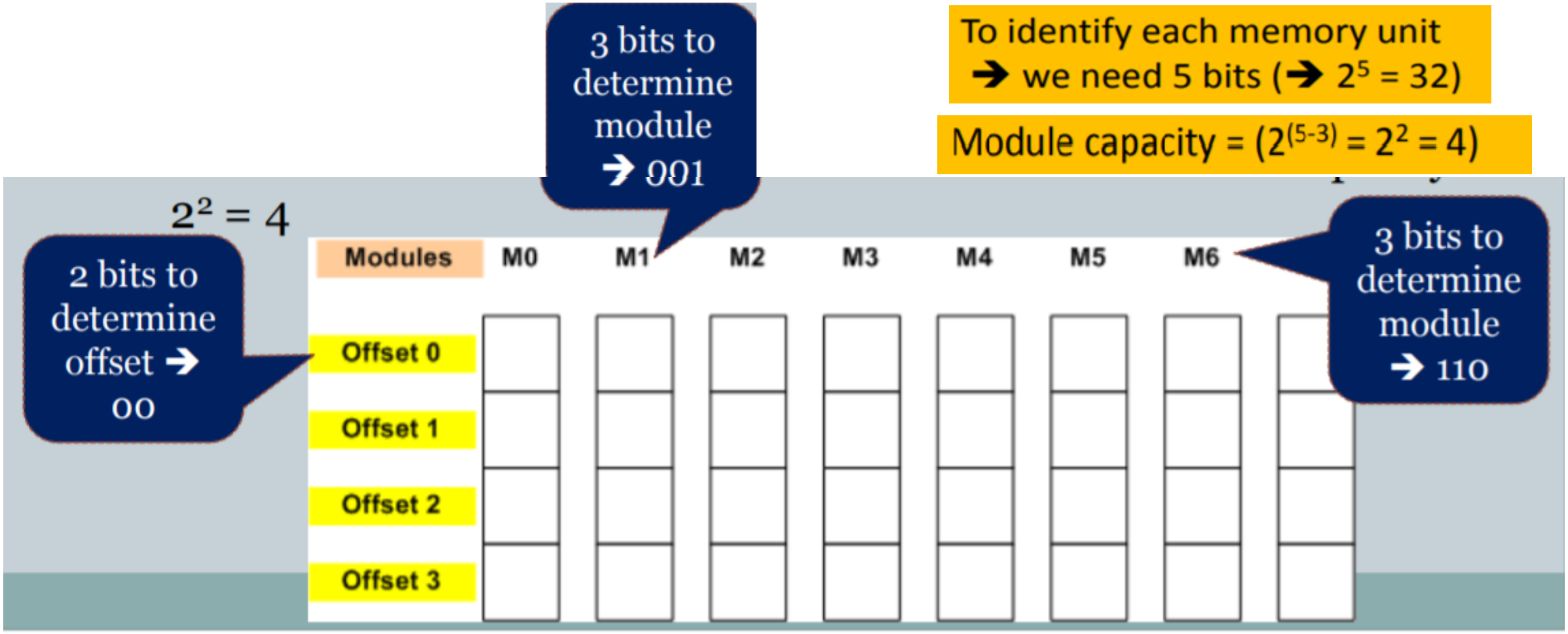
module capacity = $2^{(5-3)} = 2^2 = 4$

Memory Interleaving

Given 8 modules
➔ we need 3 bits (➔ $2^3 = 8$)

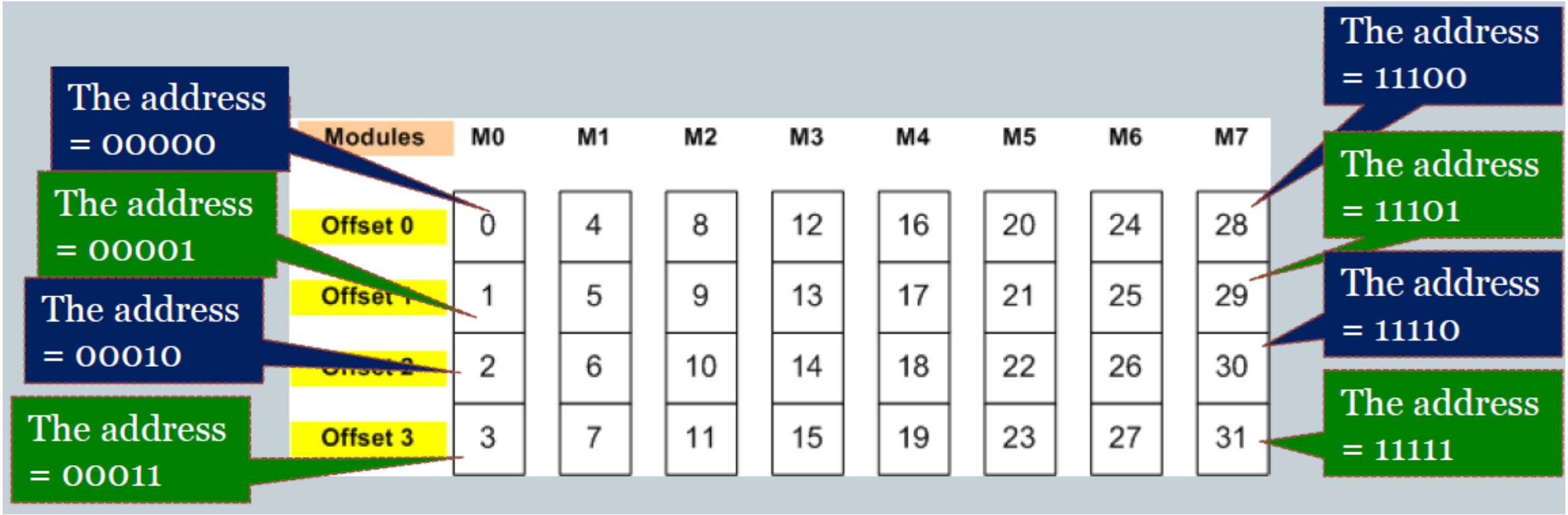
To identify each memory unit
➔ we need 5 bits (➔ $2^5 = 32$)

Module capacity = ($2^{(5-3)} = 2^2 = 4$)



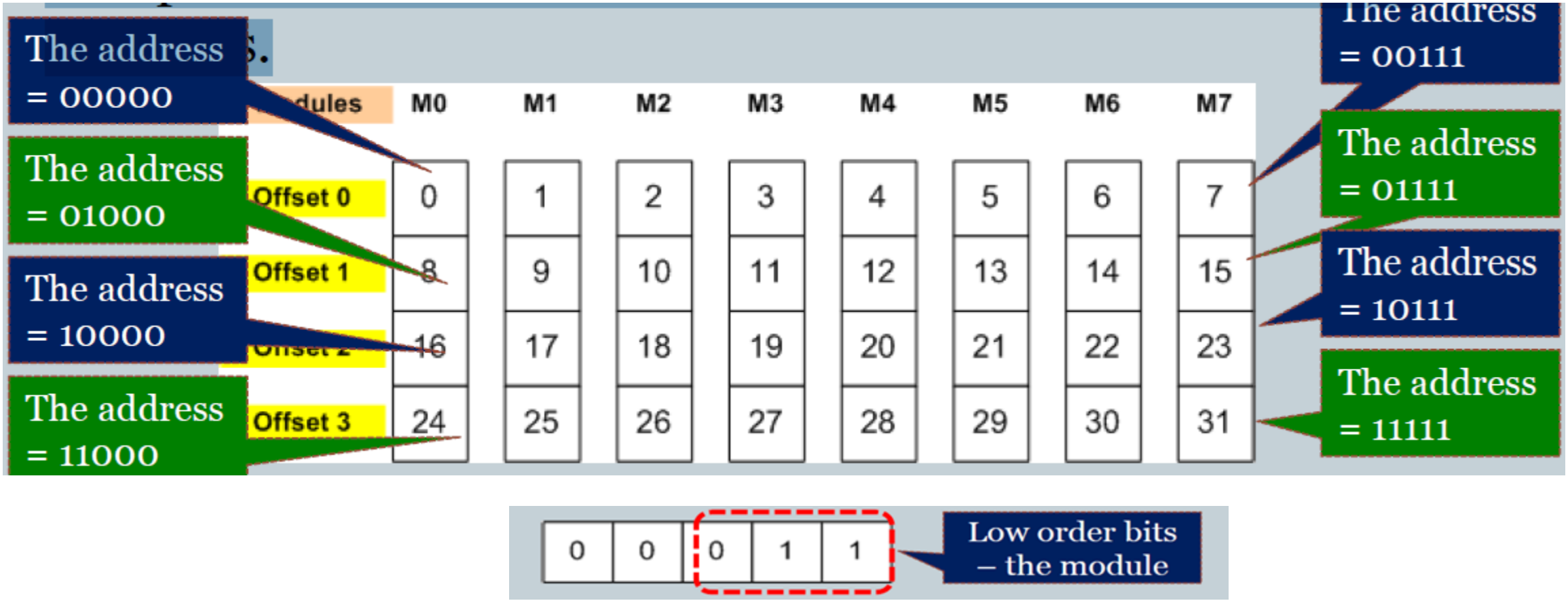
Memory Interleaving: HOI

- High Order Interleaving distributes the addresses so that each module contain consecutive addresses
- Address 3 , in binary (remember we need 5 bits) = 00011
- So, address 3 is at module 0, offset 3



Memory Interleaving: LOI

- LOI places the consecutive address in different memory modules.
- Address 3 , in binary (remember we need 5 bits) = 00011
- So, address 3 is at module 3, offset 0



Example 2

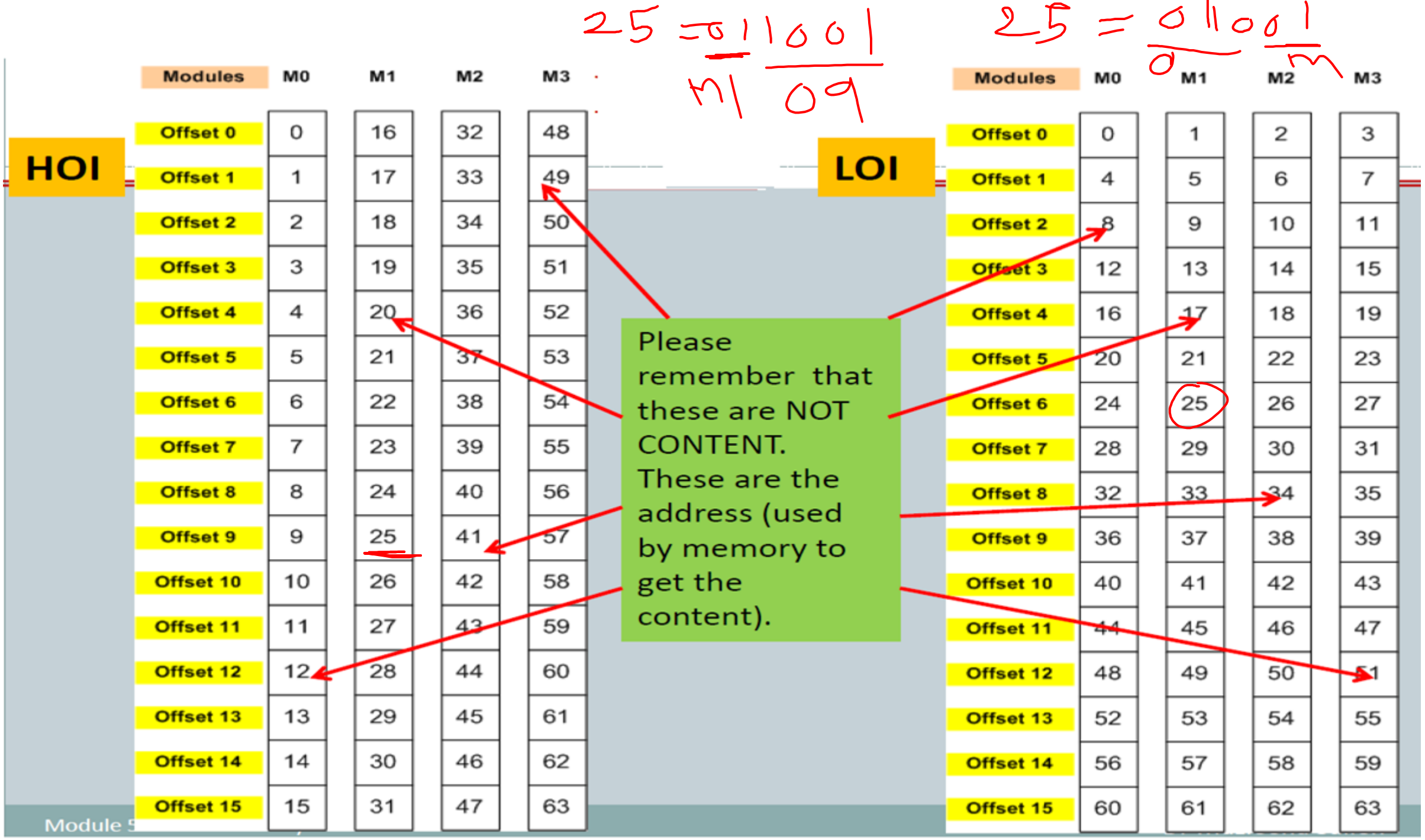
Given a memory capacity = 64 and a total main module of 4. Determine the module capacity Where can addresses 25, 32 and 55 be found in HOI and LOI? Draw the HOI and LOI memory interleaving.

-> To identify each memory unit

we need 6 bits ($2^6 = 64$)

Given 4 modules we need 2 bits ($2^2 = 4$)

Module capacity = ($2^{(6-2)} = 2^4 = 16$)


25 = 01100 1
M 09
25 = 0 1100 1
0 M
HOI
LOI
Modules
M0
M1
M2
M3
Offset 0 0 16 32 48
Offset 1 1 17 33 49
Offset 2 2 18 34 50
Offset 3 3 19 35 51
Offset 4 4 20 36 52
Offset 5 5 21 37 53
Offset 6 6 22 38 54
Offset 7 7 23 39 55
Offset 8 8 24 40 56
Offset 9 9 25 41 57
Offset 10 10 26 42 58
Offset 11 11 27 43 59
Offset 12 12 28 44 60
Offset 13 13 29 45 61
Offset 14 14 30 46 62
Offset 15 15 31 47 63
Please remember that these are NOT CONTENT. These are the address (used by memory to get the content).
Modules
M0
M1
M2
M3
Offset 0 0 1 2 3
Offset 1 4 5 6 7
Offset 2 8 9 10 11
Offset 3 12 13 14 15
Offset 4 16 17 18 19
Offset 5 20 21 22 23
Offset 6 24 25 26 27
Offset 7 28 29 30 31
Offset 8 32 33 34 35
Offset 9 36 37 38 39
Offset 10 40 41 42 43
Offset 11 44 45 46 47
Offset 12 48 49 50 51
Offset 13 52 53 54 55
Offset 14 56 57 58 59
Offset 15 60 61 62 63
Module 5

HOI:

Address 25 (011001)

➔ module 1, offset 9

Address 32(100000)

➔module 2, offset 0

Address 55 (110111)

➔ module 3, offset 7

LOI:

Address 25 (011001)

➔ module 1, offset 6

Address 32(100000)

➔module 0, offset 8

Address 55 (110111)

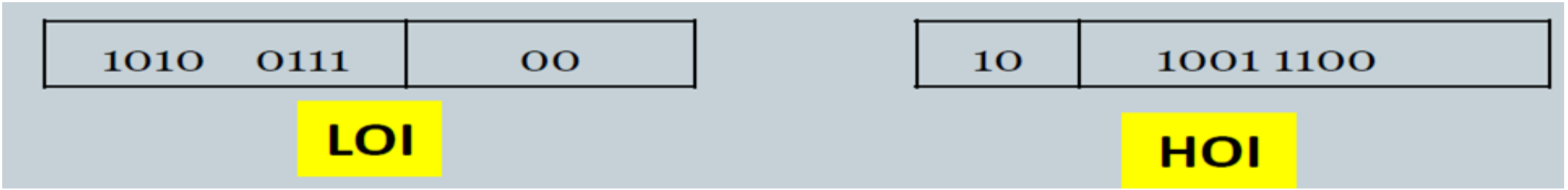
➔ module 3, offset 13

Example 3

Given a memory address as 29Ch (10 bits) and there are 4 memory banks/modules. Determine the memory bank/module address and the address of the word in the bank/module, for both HOI and LOI.

-> Given the memory address of 10 bits, 29Ch represented as 1010011100(668 in dec) in binary. Given 4 modules we need 2 bits ($2^2 = 4$)

Module capacity = $(2^{(10-2)}) = 2^8 = 256$ So 8 bits is used for this



- Memory bank/module address = **00**
- Address of the word in the bank/module = 1010 0111 = **A7h**

- Memory bank/module address = **10**
- Address of the word in the bank/module = 1001 1100 = **9Ch**

Example 4

- A main memory has 32 Mwords. There are 16 memory banks (modules). Draw the modular memory address format if the system is implemented with HOI.

To identify each 32M memory
➔ we need 25 bits (➔ $2^5 * 2^{20} = 32M$)

Given 16 modules ➔ we need 4 bits (➔ $2^4 = 16$)

Module capacity = ($2^{(25-4)} = 2^{21} = 16$)
➔ We will use 21 bits for offset

Advantages and Disadvantages of LOI

Advantages

It produces memory interference.

LOI allows for concurrent access of data stored sequentially in memory

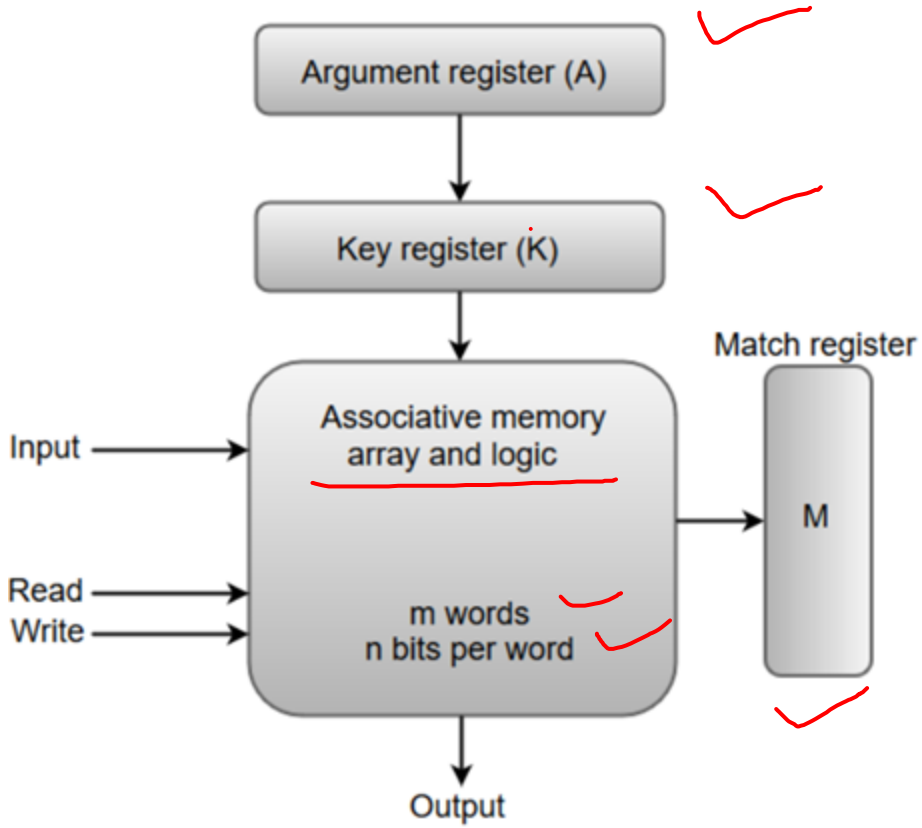
Disadvantages

A failure of any single module would be catastrophic to the whole system.

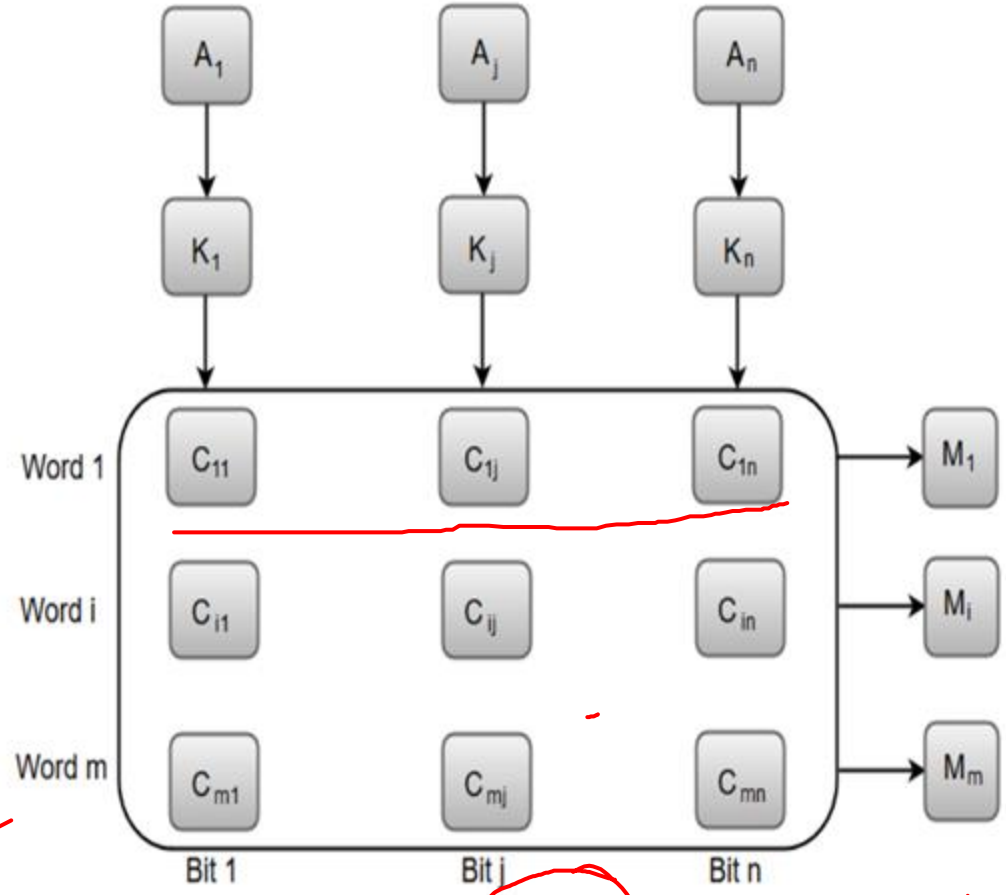
Associative Memory

- An associative memory can be considered as a memory unit whose stored data can be identified for access by the content of the data itself rather than by an address or memory location.
- Associative memory is often referred to as **Content Addressable Memory (CAM)**.
- When a write operation is performed on associative memory, no address or memory location is given to the word. The memory itself is capable of finding an empty unused location to store the word.
- when the word is to be read from an associative memory, the content of the word, or part of the word, is specified. The words which match the specified content are located by the memory and are marked for reading.

- From the block diagram, we can say that an associative memory consists of a memory array and logic for 'm' words with 'n' bits per word.
- The functional registers like the argument register **A** and key register **K** each have **n** bits, one for each bit of a word. The match register **M** consists of **m** bits, one for each memory word.
- The words which are kept in the memory are compared in parallel with the content of the argument register.
- The key register (K) provides a mask for choosing a particular field or key in the argument word. If the key register contains a binary value of all 1's, then the entire argument is compared with each memory word. Otherwise, only those bits in the argument that have 1's in their corresponding position of the key register are compared. Thus, the key provides a mask for identifying a piece of information which specifies how the reference to memory is made.



- This diagram can represent the relation between the memory array and the external registers in an associative memory.
- The cells present inside the memory array are marked by the letter C with two subscripts. The first subscript gives the word number and the second specifies the bit position in the word. For instance, the cell C_{ij} is the cell for bit **j** in word **i**.
- A bit A_j in the argument register is compared with all the bits in column **j** of the array provided that $K_j = 1$. This process is done for all columns **j** = 1, 2, 3....., n.
- If a match occurs between all the unmasked bits of the argument and the bits in word **i**, the corresponding bit M_i in the match register is set to 1. If one or more unmasked bits of the argument and the word do not match, M_i is cleared to 0.



A = 1101 1010
K = 0000 1111 → mask

word 1 = 0101 0 → A → match bits → set
word 2 = 1101 1

Thank You!!