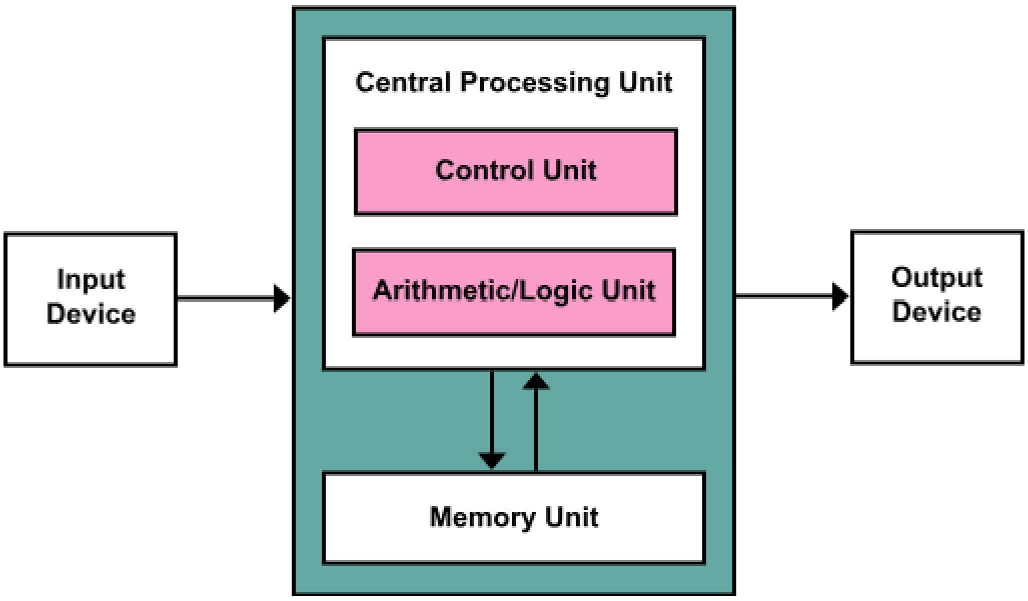
DLCA MASTER QUESTION BANK-SOLUTIONS

Theory

Q.1] Draw Detailed Von- Neumann architecture and explain in brief. Explain the advantages and disadvantages of the Von Neumann architecture. **05**

Ans:- Part A:-



Von Neumann Architecture – Detailed Explanation

The **Von Neumann architecture** is a computer design introduced by **John Von Neumann in 1945**. It is one of the most important models in computer science and is used as the base for most general-purpose computers. In this architecture, **data and instructions are stored together in the same memory** and use the **same communication system (bus)** to reach the CPU. Because of this shared design, the architecture becomes simple, flexible, and cost-effective.

The architecture is built around **five main components**:

1. **Input Unit** – Takes data and instructions from external devices and sends them into the computer for processing.
2. **Output Unit** – Sends processed results from the computer to display devices like monitors, printers, or storage.
3. **Memory Unit** – Stores both data and instructions in the same place. This memory is divided into small locations, each having its own address.
4. **Control Unit (CU)** – Reads instructions from memory, interprets them, and controls all operations inside the computer.
5. **Arithmetic and Logic Unit (ALU)** – Performs all arithmetic operations (like addition, subtraction) and logical operations (like comparison).

The CPU (ALU + CU) works by following a simple cycle called the **Fetch–Decode–Execute cycle**.

- During **Fetch**, the instruction is brought from memory to the CPU.

- During **Decode**, the CPU interprets what the instruction means.
- During **Execute**, the CPU performs the needed operation, like calculation or data movement.

All these operations use the same bus, which is why the architecture is simple but can sometimes become slow.

Part B:-

The **Von Neumann architecture** is a computer design in which **data and instructions are stored together in the same memory**. The CPU uses the **same bus** to read both instructions and data. This model is used in most general-purpose computers. Because of its simple and uniform structure, it became the basis of modern computing.

Advantages of Von Neumann Architecture (7 Points)

1. **Simple and clear design**
   The computer is easy to build because both instructions and data are kept in the same memory. This reduces wiring and makes the system easier to understand.
2. **Low hardware cost**
   Only one memory and one bus are needed, so the overall hardware becomes cheaper. Earlier computers preferred this as it reduced the cost of construction.
3. **Easy to program and operate**
   Since instructions are stored in the same memory as data, programs can be easily written, modified, and loaded. This makes compiling and debugging much simpler.
4. **Efficient use of available memory**
   Memory can be shared for both data and instructions. The system can decide how to divide memory based on what the program needs at that moment.
5. **Widely used and well supported**
   Most computers follow this architecture, so it has a lot of resources, software, and documentation. This makes development and maintenance easier.
6. **Allows self-modifying programs**
   Since programs are stored like data, a program can modify its own instructions during execution, which can be useful for certain advanced tasks.
7. **Uniform handling of data and instructions**
   Both data and instructions are treated in the same way, making the CPU design simpler and improving the consistency of the system.

Disadvantages of Von Neumann Architecture (7 Points)

1. **Von Neumann bottleneck**
   Only one instruction or data item can move through the bus at a time, which slows down the entire system.

2. **Slower execution speed**
   The CPU must fetch instructions and data one after the other, so it cannot work at full speed.
3. **Memory conflicts**
   Since data and instructions share the same memory, data can accidentally overwrite instructions and cause errors.
4. **Security risks**
   Attackers can change data in memory and make the system treat it as instructions, leading to problems like code injection.
5. **No parallel fetching**
   The architecture does not allow the CPU to fetch instructions and data at the same time, reducing performance.
6. **High CPU waiting time**
   The CPU spends a lot of time waiting for data to arrive through the single bus, which reduces efficiency.
7. **Needs additional techniques to improve speed**
   To overcome its limitations, systems must use caching, pipelining, and other techniques, making the design more complex.

Q.2] Explain bus arbitration and the different methods of centralized bus arbitration. 10

Ans:-

- None of the processors can proceed further. Such a situation is called as **Dead Lock.**

New Syll. : MU : Dec. 22

6.13.1 Bus Arbitration :

- Each processor runs independently in a loosely coupled system.
- Also there is no direct connection between the processor.
- Interprocessor is achieved with the help of shared resources. If more than one processor grants access to the common resources at the same time bus contention occurs.

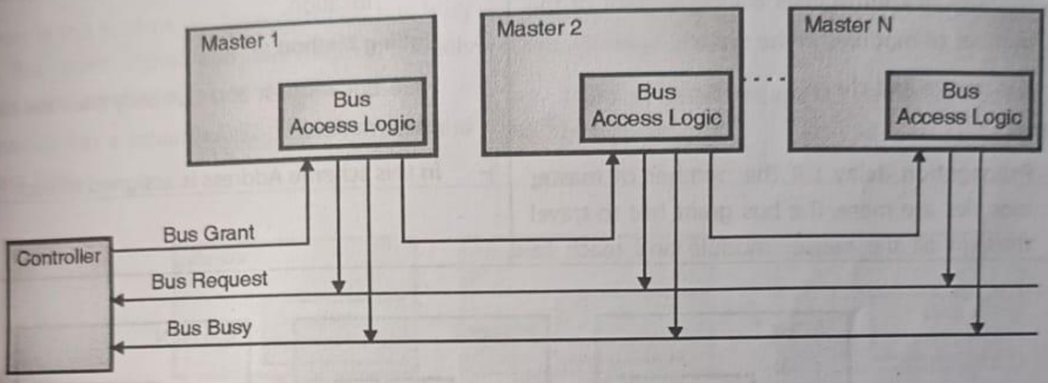


Fig. 6.13.1 : Daisy chaining

- In order to solve the problem of bus contention we use Bus arbitration.
- The different schemes of bus arbitration are :
  1. Daisy chaining.
  2. Polling method.
  3. Independent request.

1. **Daisy chaining :**

- Daisy chain method is characterized by its simplicity and low cost.
- Mainly three buses are there :
  (a) Bus request.
  (b) Bus grant.
  (c) Bus busy.
- Here Bus busy and Bus request is common.
- All masters use the same line for making bus requests.
- In response to bus request, the controller sends out a bus grant signal.
- Bus grant is activated if and only if bus busy signal is inactive.
- As shown in Fig. 6.13.1, bus grant serially propagates through each master until it gets first one master that has requested for the bus.
- Requested module receives bus grant signal, activates busy line, and gains control of the bus.
- Therefore any other module will not receive the grant signal.

How priority of module is decided ?

- Priority is determined by physical locations of the modules. The one located closest to the controller has the highest priority.
- Let's consider an example, say master 4 has made request.
- In response to this, controller will provide bus grant, if bus busy is not active.
- Now bus grant will propagate through master 1, master 2 and so on.

- But let's say till it reaches master 4, master 3 wants to gain control of the bus and at that instant it receives bus grant signal, therefore it will immediately gain the control of the bus.
- Master 4 will again wait, till master 3 releases bus.

Advantages :

1. Compared to remaining two system, daisy chain requires less number of control lines. Number of control lines is independent of the number of modules in the system.
2. It is simple and cheap.

Disadvantages :

1. **Propagation delay :** If the number of master modules are more, the bus grant has to travel through all the master module and reach to last master module, which takes lot of time. The propagation delay depends upon number of modules. If user want fast system he/she has to limit number of modules.
2. If in between module fails because of some reason, the chain is broken. This particular aspect is not allowed in multiprocessing system.
3. The priority of master is fixed by its physical location.

2. Polling Method :

- Here bus request and bus busy lines are common to each module.
- In this scheme Address is assigned to each module.

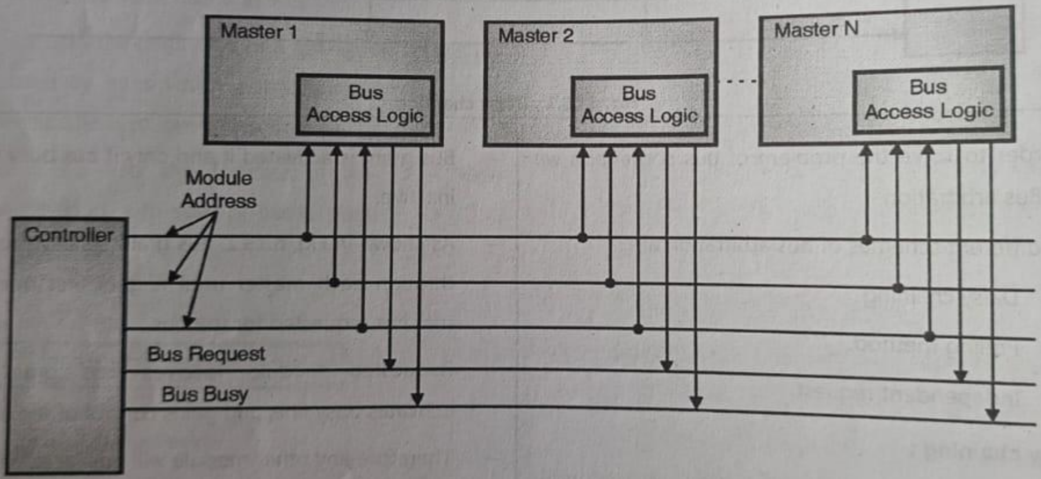


Fig. 6.13.2 : Polling method

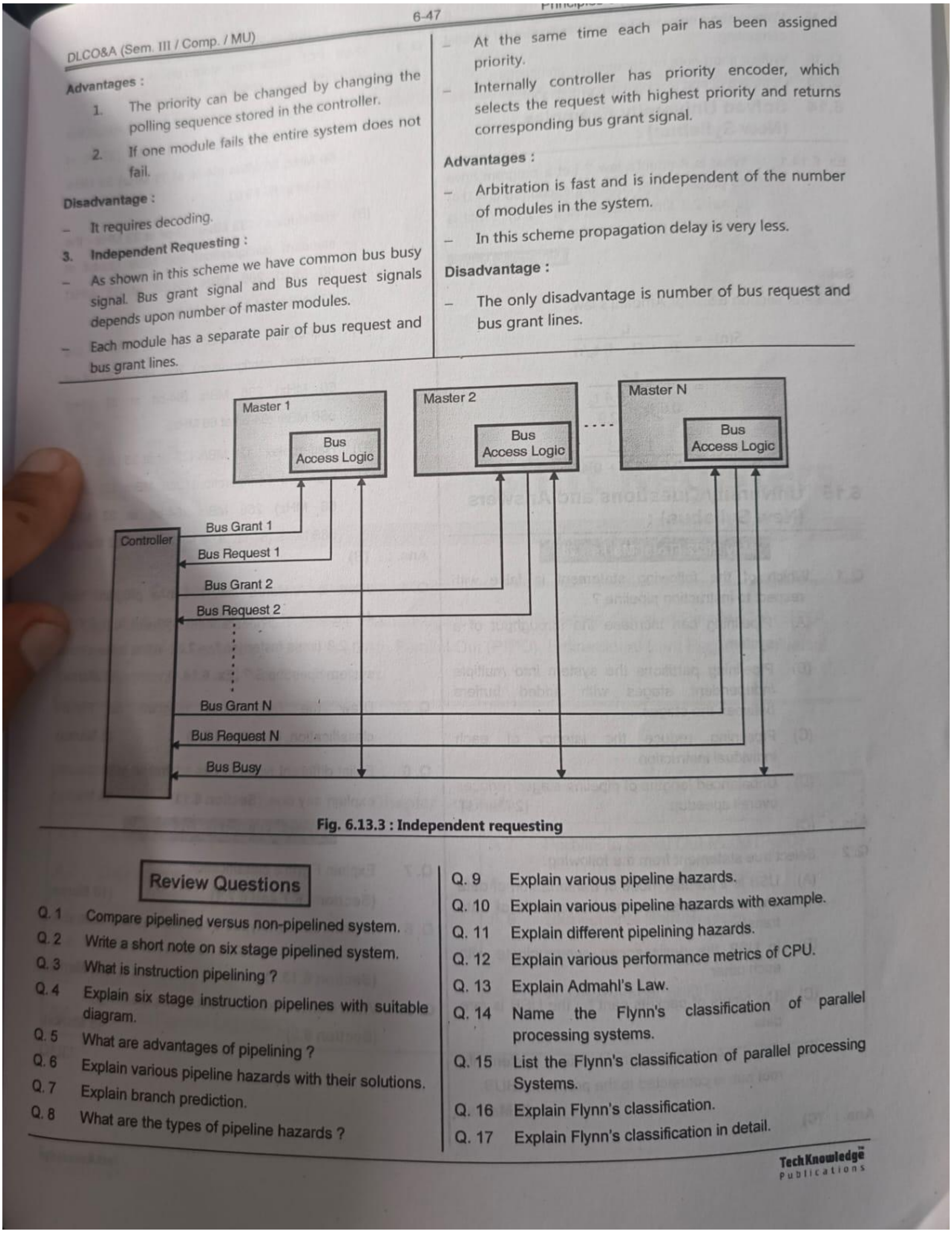
- When bus request is present, controller will generate sequence of module addresses.
- This address will be polled by each master module.
- If match occurs that particular module will activate busy line and begins to use bus.

What about priority ?

- The address which is outputted first by controller, gets first priority.
- Here the main advantage of polling is that the priority can be dynamically changed by altering the polling sequence stored in the controller.

What about propagation delay ?

- Propagation delay depends upon number of master modules.
- If number of modules are more polling will take more time.



Advantages :

1. The priority can be changed by changing the polling sequence stored in the controller.
2. If one module fails the entire system does not fail.

Disadvantage :

– It requires decoding.

3. **Independent Requesting :**

– As shown in this scheme we have common bus busy signal. Bus grant signal and Bus request signals depends upon number of master modules.

– Each module has a separate pair of bus request and bus grant lines.

– At the same time each pair has been assigned priority.

– Internally controller has priority encoder, which selects the request with highest priority and returns corresponding bus grant signal.

Advantages :

– Arbitration is fast and is independent of the number of modules in the system.

– In this scheme propagation delay is very less.

Disadvantage :

– The only disadvantage is number of bus request and bus grant lines.

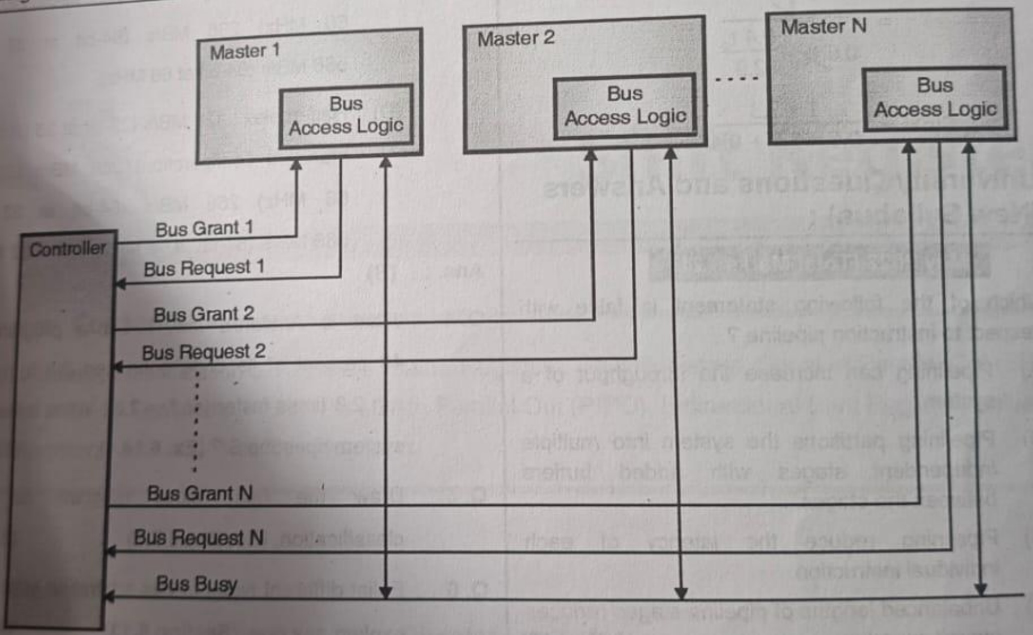


Fig. 6.13.3 : Independent requesting

Review Questions

Q. 1 Compare pipelined versus non-pipelined system.
Q. 2 Write a short note on six stage pipelined system.
Q. 3 What is instruction pipelining ?
Q. 4 Explain six stage instruction pipelines with suitable diagram.
Q. 5 What are advantages of pipelining ?
Q. 6 Explain various pipeline hazards with their solutions.
Q. 7 Explain branch prediction.
Q. 8 What are the types of pipeline hazards ?
Q. 9 Explain various pipeline hazards.
Q. 10 Explain various pipeline hazards with example.
Q. 11 Explain different pipelining hazards.
Q. 12 Explain various performance metrics of CPU.
Q. 13 Explain Admahl's Law.
Q. 14 Name the Flynn's classification of parallel processing systems.
Q. 15 List the Flynn's classification of parallel processing Systems.
Q. 16 Explain Flynn's classification.
Q. 17 Explain Flynn's classification in detail.

TechKnowledge Publications

Q.3] Explain the various addressing modes. **05**

Ans:-

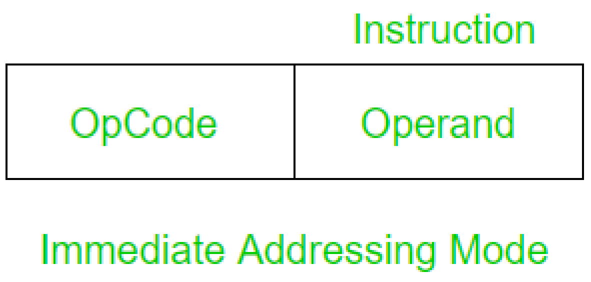
Definition of Addressing Modes (4 points)

1. Addressing modes are the different ways a CPU finds the location of data needed for an instruction.
2. They tell the processor **where the operand is stored** (inside the instruction, in a register, or in memory).
3. These modes make instructions flexible and easier to use in different situations.
4. They help in writing efficient programs, handling arrays, stacks, jumps, and memory access.
   Example: MOV A, 2000H → here 2000H is the address of data depending on the addressing mode.

---

1. Immediate Addressing Mode (3 points + example)

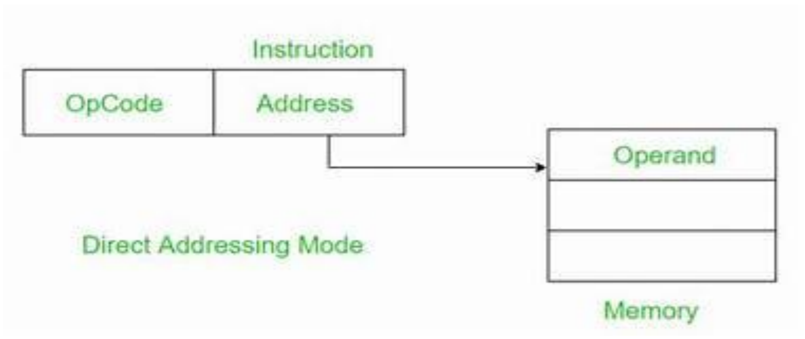
1. The data (operand) is given **directly inside the instruction**.
2. No memory access is needed because the value is already provided.
3. It is the fastest addressing mode because data is available immediately.
   Example: MOV A, #10 → the value 10 is directly given in the instruction.



---

2. Direct Addressing Mode (3 points + example)

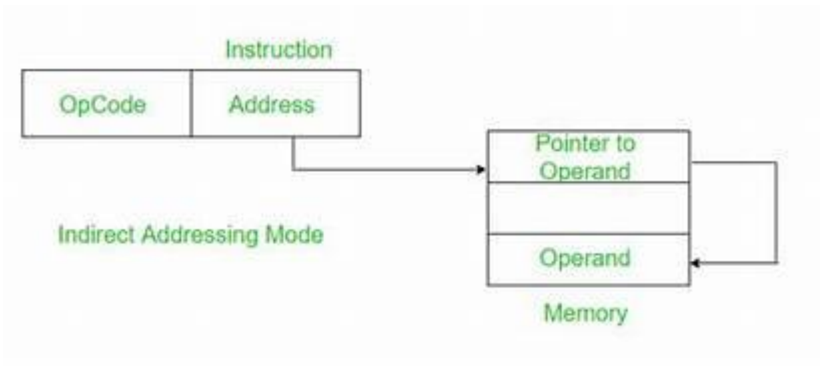
1. The instruction contains the **actual memory address** of the data.
2. The CPU goes directly to that memory location to fetch the data.
3. It is simple to understand but slower than immediate mode because memory access is needed.
   Example: MOV A, 2050H → data is taken from memory address 2050H.



---

3. Indirect Addressing Mode (3 points + example)

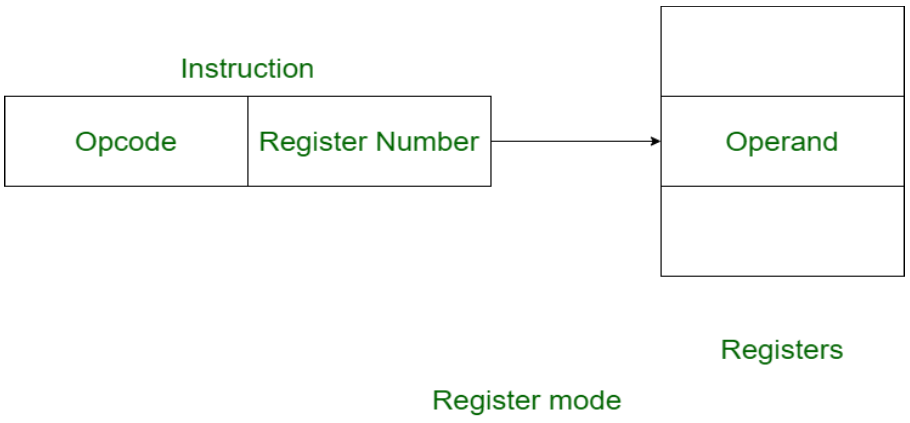
1. The instruction contains the **address of a memory location**, which holds the real address of the data.
2. The CPU performs **two steps**: first fetch the address, then fetch the data.
3. It provides more flexibility in accessing memory.
   Example: MOV A, @5000H → 5000H contains another address where the actual data is stored.



---

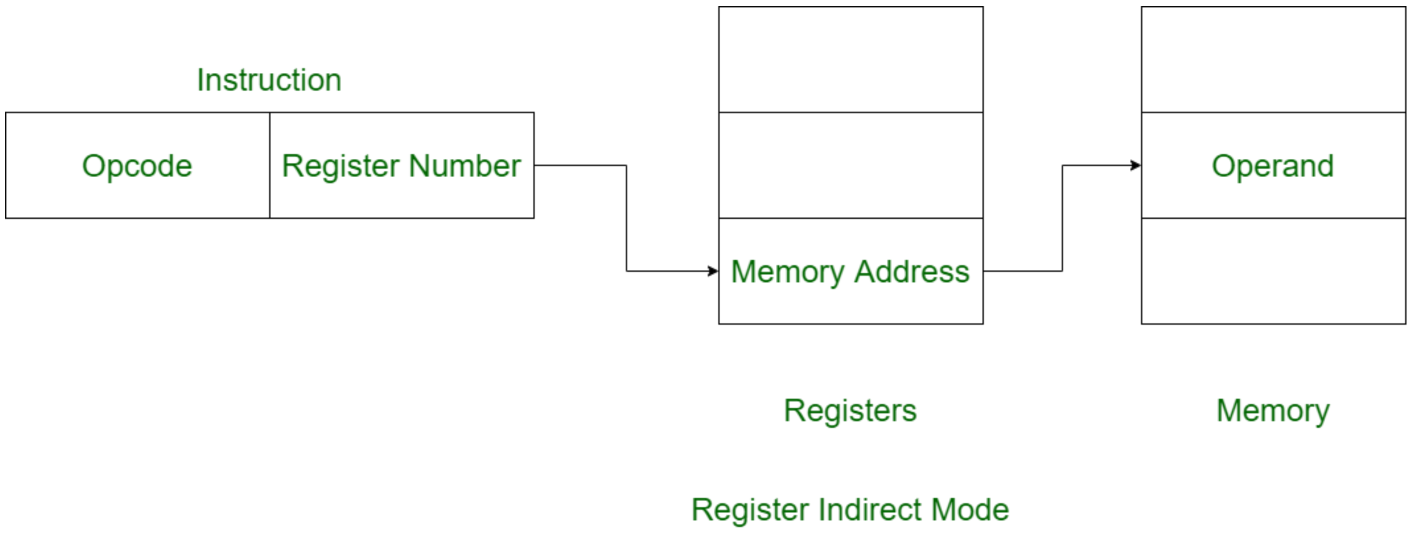
4. Register Addressing Mode (3 points + example)

1. The data is stored inside a **CPU register**, and the instruction specifies the register.
2. Register access is very fast because registers are inside the CPU.
3. It is commonly used in arithmetic and logical operations.
   Example: MOV A, R1 → data is taken from register R1.



5. Register Indirect Addressing Mode (3 points + example)

1. A register contains the **memory address** of the data.
2. The CPU uses the register's value as the address and fetches the data from memory.
3. Faster than indirect addressing because the address is stored in a register.
   Example: MOV A, @R0 → R0 contains the address where data is stored.



6. Displacement Addressing Mode (Indexed/Base + Offset) (3 points + example)

1. The final address is obtained by adding a **constant value (offset)** to a register value.
2. Used for accessing arrays, tables, and structured data.
3. It provides flexible and easy access to sequential memory locations.
   Example: MOV A, 10(R1) → final address = R1 + 10.

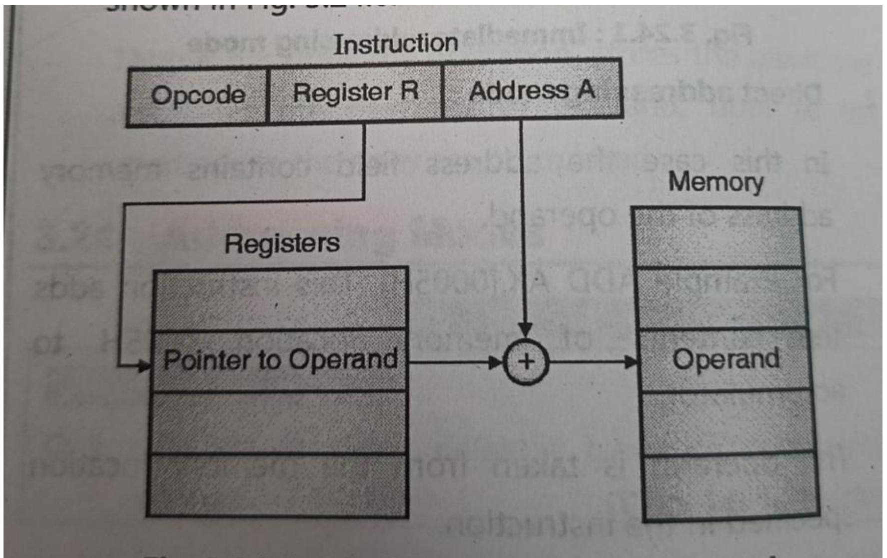


Fig. 3.24.6 : Displacement addressing mode

---

7. Relative Addressing Mode (3 points + example)

1. The address of the operand is determined by adding an **offset** to the **program counter (PC)**.
2. Mostly used for jumps and branches within a program.
3. Makes it easy to move to nearby instructions in a code block.
   Example: JMP +5 → jump to the instruction 5 steps ahead of the current one.

---

8. Stack Addressing Mode (3 points + example)

1. Operands are taken from the **stack**, using LIFO (Last In First Out).
2. The stack pointer (SP) automatically moves up or down during push or pop operations.
3. Used for subroutine calls, expression evaluation, and temporary storage.
   Example: POP A → removes (pops) the top value from the stack into A.

Q.4] Explain how NAND is a universal logic gate with examples.

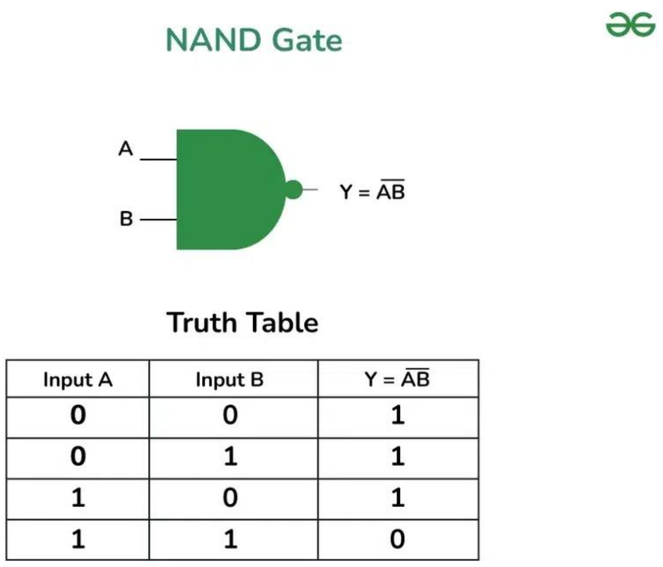
Short Note on Logic Gate. **10**

Ans:-Part A:-

The universal gates are the logic gates that are versatile in that they can be programmed to execute any Boolean function. The most popular types among them are NAND and NOR gates.

To explain it further, the input of the universal gates can be interconnected in one way or the other while the inputs can also be inverted in one way or the other to get the AND, OR, NOT, XOR and XNOR basic Boolean functions.

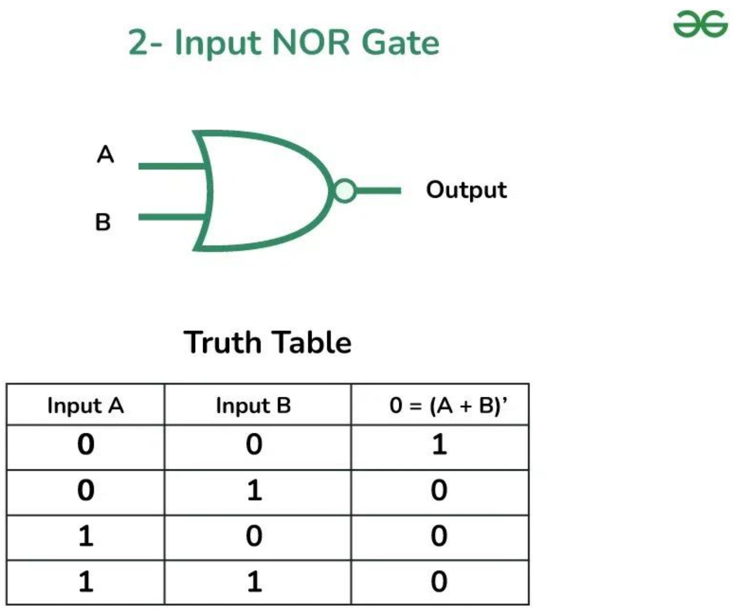
They are very versatile to be core components when it comes to the implementation of digital circuits and processors. By employing just a single gate or by combining both these gate types, it is possible to construct intricate logic systems.



Input A	Input B	$Y = \overline{AB}$
0	0	1
0	1	1
1	0	1
1	1	0

NAND Gate

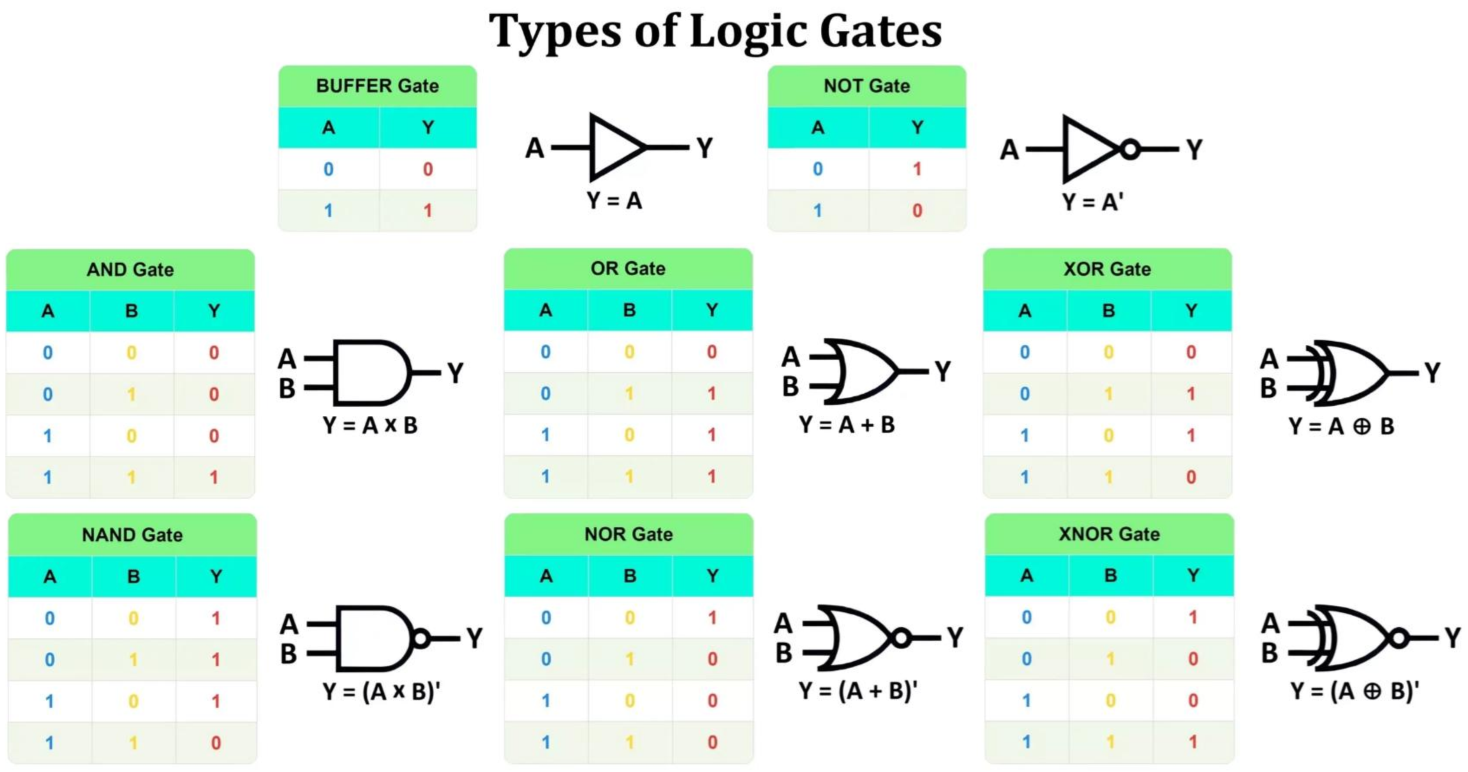
A NAND gate is a combination of an AND gate followed by a NOT gate. It outputs a 0 only when all its inputs are 1; otherwise, it outputs 1.



Input A	Input B	0 = (A + B)’
0	0	1
0	1	0
1	0	0
1	1	0

NOR Gate

A NOR gate is a combination of an OR gate followed by a NOT gate. It outputs a 1 only when all its inputs are 0; otherwise, it outputs 0.



BUFFER Gate

A	Y
0	0
1	1

NOT Gate

A	Y
0	1
1	0

AND Gate

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

OR Gate

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

XOR Gate

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

NAND Gate

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

NOR Gate

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

XNOR Gate

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

Definition

A logic gate is called **universal** if you can build **all other basic gates** (NOT, AND, OR, NOR, XOR, XNOR) using only that gate.
NAND gate is universal because **any digital circuit can be made using only NAND gates**.

Why NAND is a Universal Gate? (4 Points)

1. **NAND can perform NOT operation**
   If you connect **both inputs of NAND** to the **same signal**, it works like a **NOT gate**.
2. **NAND can be combined to form AND gate**
   Two NAND gates together recreate the normal AND function.
3. **NAND can form OR gate using De Morgan's law**
   By first inverting inputs with NAND (acting as NOT) and then applying NAND again, it behaves like OR.
4. **Any complex circuit (Adder, Decoder, Multiplexer)** can be built using only NAND gates
   Because NAND can create all basic gates, **it becomes universal**.

Examples

1. NAND used as NOT gate

Input A is connected to both inputs:

A	**Output = A NAND A**
0	1
1	0

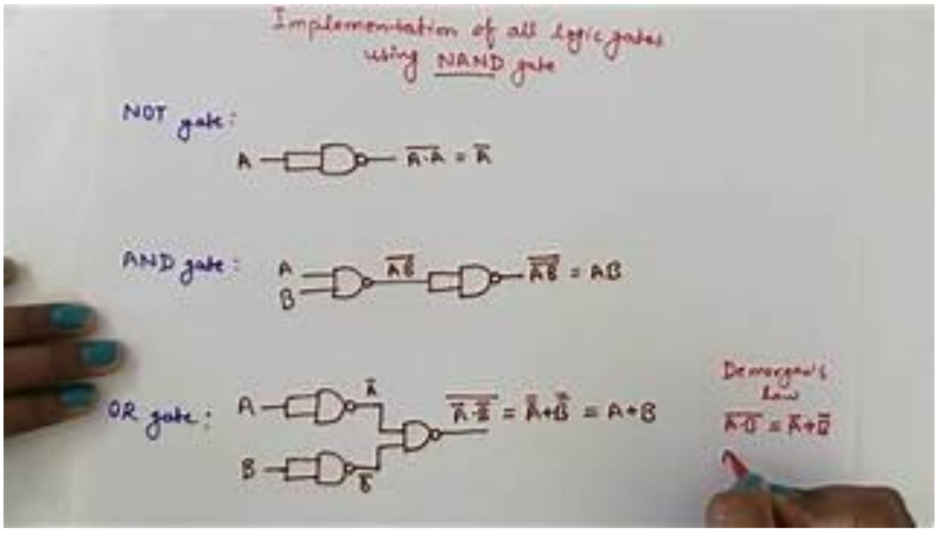
This behaves exactly like a NOT gate.

---

2. NAND used as AND gate

To build AND:

- First NAND the inputs → gives NOT(AB)
- NAND the result with itself → gives AB


Implementation of all logic gates using NAND gate
NOT gate:
A
$\overline{A \cdot A} = \overline{A}$
AND gate:
A
B
$\overline{AB}$
$\overline{\overline{AB}} = AB$
OR gate:
A
B
$\overline{A}$
$\overline{B}$
$\overline{\overline{A} \cdot \overline{B}} = \overline{\overline{A}} + \overline{\overline{B}} = A + B$
Demorgan's Law
$\overline{A \cdot B} = \overline{A} + \overline{B}$

---

3. NAND used as OR gate

Using De Morgan's theorem:

A + B = (A'B')'

Implementation:

- Use two NAND gates as NOT for A and B
- Then NAND the two outputs

Works exactly like an OR gate.

Part B:-

Short Note: Logic Gates

Logic gates are the basic building blocks of digital circuits. They perform logical operations on one or more binary inputs to produce a single binary output. Each logic gate follows a specific truth table that defines how the output changes with respect to the inputs. Logic gates are used in computers, calculators, digital watches, and all electronic devices that process binary data.

The important logic gates are:

1. AND Gate

Gives output **1** only if **both inputs are 1**.
It performs logical multiplication.

2. OR Gate

Gives output **1** if **any one input is 1**.
It performs logical addition.

3. NOT Gate

Also called an inverter.
It gives the **opposite** of the input:
Input 0 → Output 1
Input 1 → Output 0

4. NAND Gate

It is the NOT of AND.
Gives output 0 only when both inputs are 1.

5. NOR Gate

It is the NOT of OR.
Gives output 1 only when both inputs are 0.

6. XOR Gate (Exclusive OR)

Gives output 1 when inputs are **different**.

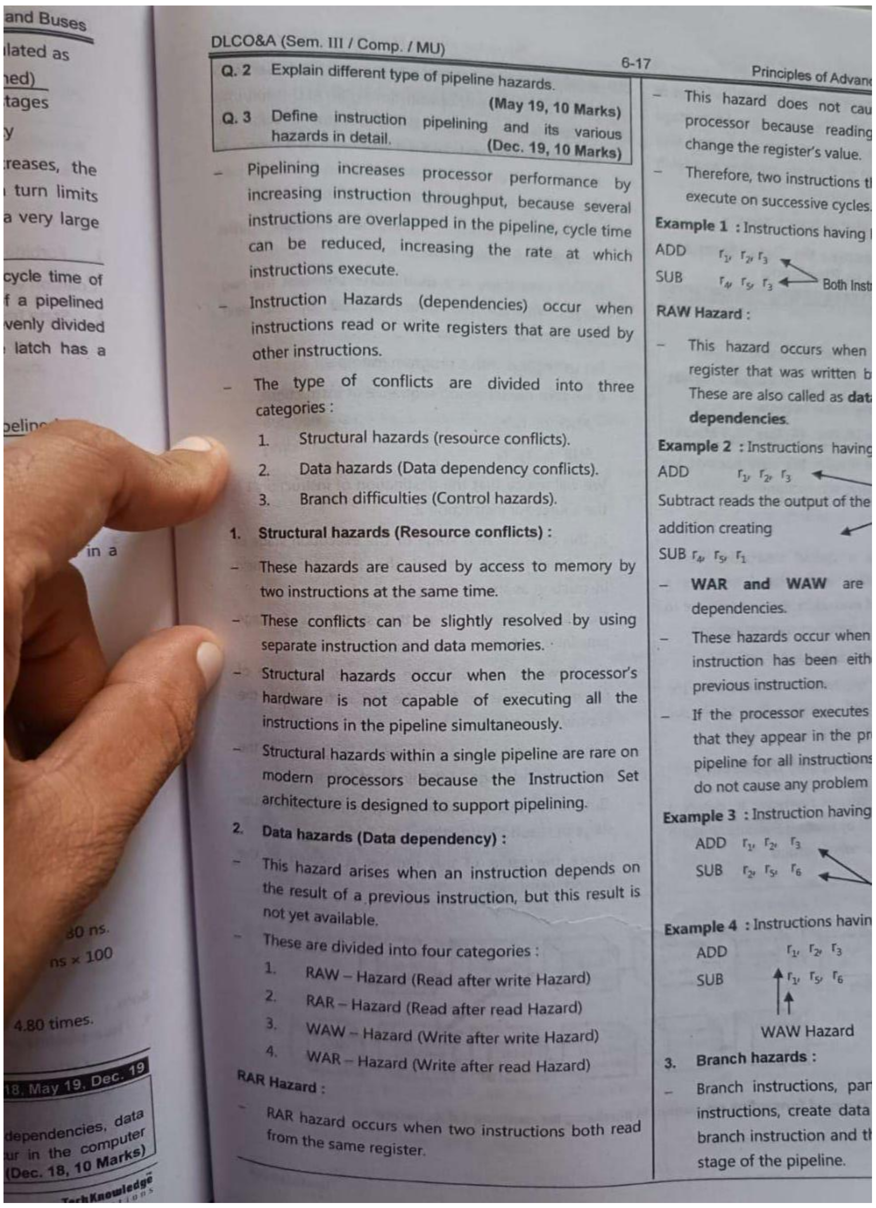
7. XNOR Gate (Exclusive NOR)

Gives output 1 when inputs are **the same**.

Logic gates are combined to design adders, multiplexers, decoders, memory units, microprocessors, and all types of digital systems. They form the foundation of all digital electronics.

Q.5] Explain classic instruction pipeline stages. What are pipeline hazards? 10

Ans:-



and Buses
lated as
ned)
tages
y
reases, the
turn limits
a very large

cycle time of
f a pipelined
venly divided
latch has a

pelin

in a

30 ns.
ns × 100

4.80 times.

18, May 19, Dec. 19

dependencies, data
ur in the computer
(Dec. 18, 10 Marks)

Q. 2 Explain different type of pipeline hazards. **(May 19, 10 Marks)**

Q. 3 Define instruction pipelining and its various hazards in detail. **(Dec. 19, 10 Marks)**

- Pipelining increases processor performance by increasing instruction throughput, because several instructions are overlapped in the pipeline, cycle time can be reduced, increasing the rate at which instructions execute.
- Instruction Hazards (dependencies) occur when instructions read or write registers that are used by other instructions.
- The type of conflicts are divided into three categories :
  1. Structural hazards (resource conflicts).
  2. Data hazards (Data dependency conflicts).
  3. Branch difficulties (Control hazards).

1. Structural hazards (Resource conflicts) :

- These hazards are caused by access to memory by two instructions at the same time.
- These conflicts can be slightly resolved by using separate instruction and data memories.
- Structural hazards occur when the processor's hardware is not capable of executing all the instructions in the pipeline simultaneously.
- Structural hazards within a single pipeline are rare on modern processors because the Instruction Set architecture is designed to support pipelining.

2. Data hazards (Data dependency) :

- This hazard arises when an instruction depends on the result of a previous instruction, but this result is not yet available.
- These are divided into four categories :
  1. RAW – Hazard (Read after write Hazard)
  2. RAR – Hazard (Read after read Hazard)
  3. WAW – Hazard (Write after write Hazard)
  4. WAR – Hazard (Write after read Hazard)

RAR Hazard :

- RAR hazard occurs when two instructions both read from the same register.

- This hazard does not cau processor because reading change the register's value.
- Therefore, two instructions t execute on successive cycles.

Example 1 : Instructions having

ADD r_1, r_2, r_3
SUB r_4, r_5, r_3 Both Instr

RAW Hazard :

- This hazard occurs when register that was written b These are also called as **dat dependencies.**

Example 2 : Instructions having

ADD r_1, r_2, r_3
Subtract reads the output of the addition creating
SUB r_4, r_5, r_1

- **WAR and WAW** are dependencies.
- These hazards occur when instruction has been eith previous instruction.
- If the processor executes that they appear in the pr pipeline for all instructions do not cause any problem

Example 3 : Instruction having

ADD r_1, r_2, r_3
SUB r_2, r_5, r_6

Example 4 : Instructions havin

ADD r_1, r_2, r_3
SUB r_1, r_5, r_6

WAW Hazard

3. Branch hazards :

- Branch instructions, par instructions, create data branch instruction and t stage of the pipeline.

type of pipeline hazards.
(May 19, 10 Marks)
ion pipelining and its various
.
(Dec. 19, 10 Marks)

ses processor performance by
tion throughput, because several
erlapped in the pipeline, cycle time
increasing the rate at which
te.

rds (dependencies) occur when
or write registers that are used by

onflicts are divided into three

azards (resource conflicts).

ds (Data dependency conflicts).

iculties (Control hazards).

ls (Resource conflicts) :

re caused by access to memory by
at the same time.

can be slightly resolved by using
ion and data memories.

rds occur when the processor's
ot capable of executing all the
he pipeline simultaneously.

ds within a single pipeline are rare on
ssors because the Instruction Set
designed to support pipelining.

Data dependency) :

rises when an instruction depends on
previous instruction, but this result is
ble.

ded into four categories :

Hazard (Read after write Hazard)

Hazard (Read after read Hazard)

– Hazard (Write after write Hazard)

– Hazard (Write after read Hazard)

d occurs when two instructions both read
ame register.

– This hazard does not cause a problem for the processor because reading a register does not change the register's value.

– Therefore, two instructions that have RAR hazard can execute on successive cycles.

Example 1 : Instructions having RAR hazard.

ADD r_1, r_2, r_3

SUB r_4, r_5, r_3

Both Instructions read r_3, creating RAR

RAW Hazard :

– This hazard occurs when an instruction reads a register that was written by a previous instruction. These are also called as **data dependencies** (or) **true dependencies**.

Example 2 : Instructions having RAW - Hazard.

ADD r_1, r_2, r_3

Subtract reads the output of the addition creating

RAW hazard

SUB r_4, r_5, r_1

– **WAR and WAW** are also called as name dependencies.

– These hazards occur when the output register of an instruction has been either read or written by a previous instruction.

– If the processor executes instructions in the order that they appear in the program and uses the same pipeline for all instructions, WAR and WAW hazards do not cause any problem in execution process.

Example 3 : Instruction having WAR Hazard.

ADD r_1, r_2, r_3

SUB r_2, r_5, r_6

WAR hazard

Example 4 : Instructions having WAW hazard

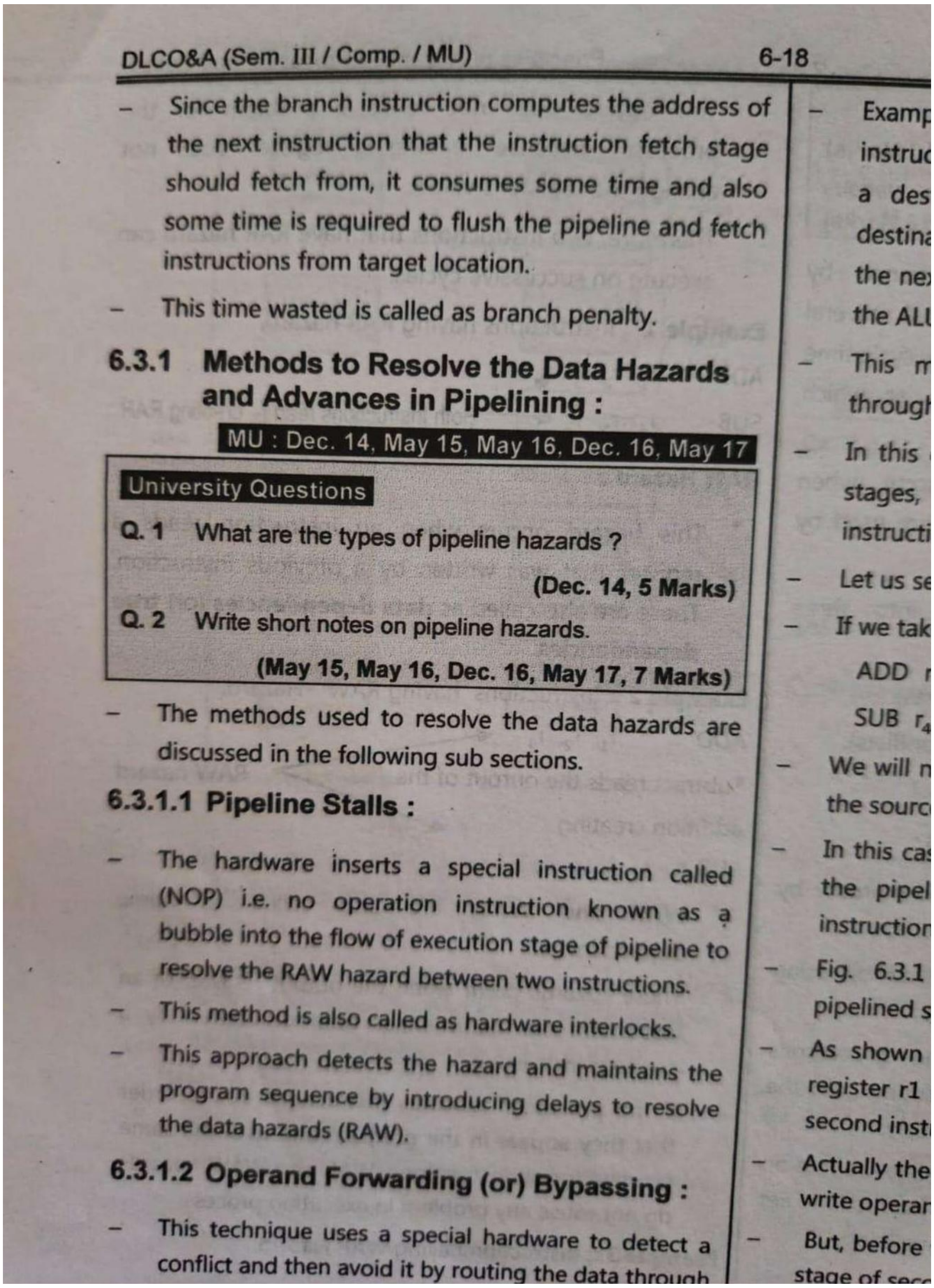
ADD r_1, r_2, r_3

SUB r_1, r_5, r_6

WAW Hazard

3. **Branch hazards :**

– Branch instructions, particularly conditional branch instructions, create data dependencies between the branch instruction and the previous instruction, fetch stage of the pipeline.



- Since the branch instruction computes the address of the next instruction that the instruction fetch stage should fetch from, it consumes some time and also some time is required to flush the pipeline and fetch instructions from target location.
- This time wasted is called as branch penalty.

6.3.1 Methods to Resolve the Data Hazards and Advances in Pipelining :

MU : Dec. 14, May 15, May 16, Dec. 16, May 17

University Questions

Q. 1 What are the types of pipeline hazards ?

(Dec. 14, 5 Marks)

Q. 2 Write short notes on pipeline hazards.

(May 15, May 16, Dec. 16, May 17, 7 Marks)

- The methods used to resolve the data hazards are discussed in the following sub sections.

6.3.1.1 Pipeline Stalls :

- The hardware inserts a special instruction called (NOP) i.e. no operation instruction known as a bubble into the flow of execution stage of pipeline to resolve the RAW hazard between two instructions.
- This method is also called as hardware interlocks.
- This approach detects the hazard and maintains the program sequence by introducing delays to resolve the data hazards (RAW).

6.3.1.2 Operand Forwarding (or) Bypassing :

- This technique uses a special hardware to detect a conflict and then avoid it by routing the data through

- Examp
instruc
a des
destina
the ne
the ALU
- This m
through
- In this
stages,
instructi
- Let us se
- If we tak
ADD r
SUB r_4
- We will n
the sourc
- In this cas
the pipel
instruction
- Fig. 6.3.1
pipelined s
- As shown
register r1
second inst
- Actually the
write operar
- But, before
stage of sec

Q.6] List methods used for designing a Hardwired Control Unit. **03**

Ans:-

Definition of Hardwired Control Unit

A **Hardwired Control Unit** is a control unit in which the control signals are generated using **fixed combinational circuits, flip-flops, gates, counters, and decoders**, instead of microprograms. It is fast because the control logic is directly implemented in hardware.

---

1. State Table Method

Definition:

A method where the behavior of the control unit is described using a **state table** that lists all states, control outputs, and next-state logic.

7 Points:

1. Designs the control unit using a full list of states.
2. Each state corresponds to a step in instruction execution.
3. Control signals are assigned to each state in the table.
4. Next-state logic is derived directly from the table.
5. Boolean functions are designed for transitions.
6. Systematic and easy to understand for small CPUs.
7. Implemented using flip-flops + combinational logic.

---

2. Delay (Timing) Method

Definition:

A method that uses **timing circuits or delay elements** to generate control signals at fixed time intervals.

7 Points:

1. Instruction execution is divided into time-based phases.
2. Delay elements produce pulses in sequence.
3. Each pulse triggers a specific micro-operation.
4. No explicit state transition logic is needed.
5. Good for simple, linear instruction sequences.
6. Hardware is easy to build but rigid.
7. Not suitable for complex branching.

---

3. Sequence Counter Method

Definition:

A method that uses a **binary counter** to step through control sequences, where each counter value represents a micro-operation step.

7 Points:

1. Counter increments every clock cycle.
2. Each counter state maps to a control step.
3. Decoder converts counter output to control signals.
4. Supports jumps by loading new counter values.
5. Simple and fast implementation.
6. Works well for sequential instructions.
7. Requires additional logic for branching.

---

4. PLA (Programmable Logic Array) Method

Definition:

A method in which a **PLA** is used to generate control signals through a programmable network of AND–OR gates.

7 Points:

1. Stores control logic in programmable form.
2. Uses AND array to detect input conditions.
3. Uses OR array to generate output signals.
4. Highly flexible and modifiable.
5. Reduces wiring complexity.
6. Good for complex processor designs.
7. Fast and compact hardware solution.

Q.7] List the levels of memory hierarchy. **05**

Ans:-

Levels of Memory Hierarchy:-

1. Registers
Registers are the fastest and smallest memory inside the CPU. They hold the data and instructions the processor is using at that exact moment. Only a few registers exist because they are very expensive and very fast.

2. Cache Memory
Cache is a fast memory placed between the CPU and RAM. It stores frequently used data so the CPU can access it quickly. Cache is divided into L1, L2, and L3 levels. L1 is smallest and fastest; L3 is larger and slightly slower.

3. Main Memory (RAM)
RAM is the primary memory where running programs and active data are stored. It is faster than storage devices but slower than cache. RAM is volatile, meaning all data is lost when power is turned off.

4. Secondary Storage – SSD
SSD stores the operating system, applications, and files. It is non-volatile and much faster than a hard disk. SSDs have no moving parts and provide quick boot and load times. They store data even when power is off.

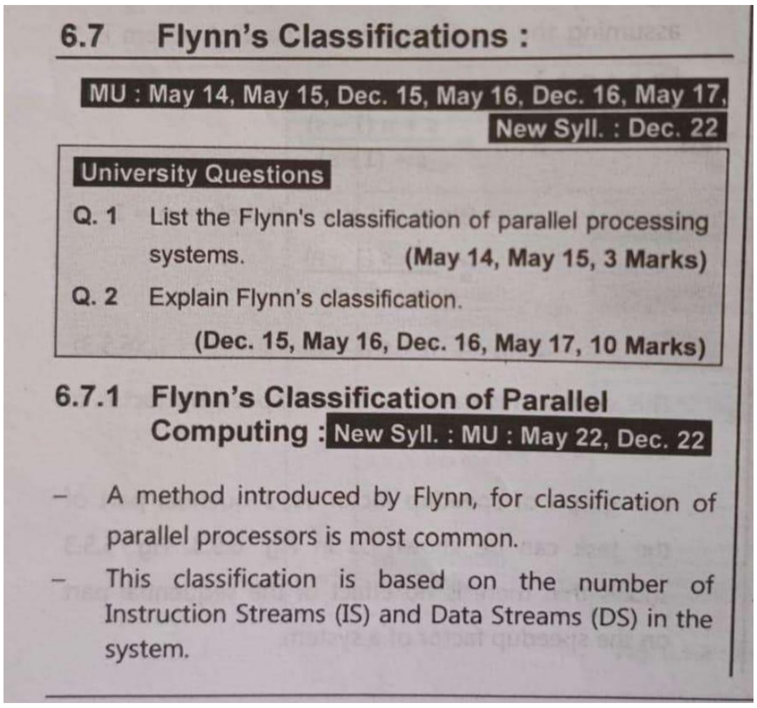
5. Secondary Storage – HDD
HDD is a magnetic storage device with large capacity. It is slower than SSD because it has mechanical moving parts. HDD is used for long-term storage of files, movies, documents, and backups.

6. Tertiary Storage – Optical or Magnetic Media
This level includes CDs, DVDs, Blu-ray discs, and magnetic tapes. These are mainly used for backup, archiving, and storing data that is not needed very often. They have slow access speeds.

7. Cloud or Remote Storage
Cloud storage is accessed through the internet. Data is stored on remote servers managed by service providers. It offers very large storage capacity but depends on network speed, so access time is slower.

Q.8] Draw and explain MIMD architecture under Flynn's classification. **10**

Ans:-



6.7 Flynn's Classifications :

MU : May 14, May 15, Dec. 15, May 16, Dec. 16, May 17, New Syll. : Dec. 22

University Questions

Q. 1 List the Flynn's classification of parallel processing systems. **(May 14, May 15, 3 Marks)**

Q. 2 Explain Flynn's classification. **(Dec. 15, May 16, Dec. 16, May 17, 10 Marks)**

6.7.1 Flynn's Classification of Parallel Computing : New Syll. : MU : May 22, Dec. 22

- A method introduced by Flynn, for classification of parallel processors is most common.
- This classification is based on the number of Instruction Streams (IS) and Data Streams (DS) in the system.

- There may be single or multiple streams of each of these.
- Hence accordingly, Flynn classified the parallel processing into four categories :

1.	Single Instruction Single Data (SISD).
2.	Single Instruction Multiple Data (SIMD).
3.	Multiple Instruction Single Data (MISD).
4.	Multiple Instruction Multiple Data (MIMD).

1. Single Instruction Single Data (SISD) :

- In this case there is a single processor that executes one instruction at a time on single data stored in the memory.
- In fact, this type of processing can be said to be unit processing, hence unit processors fall into this category.
- Fig. 6.7.1 shows this type of system. You will notice there is a Control Unit (CU) that accepts the instruction from the processor and decodes it.
- The Processing Element (PE) accesses the data from the memory and performs the operation on this data as per the signal given by control unit.
- The Memory Module (MM) is connected to the PE and the CU for the data and the instruction streams respectively.

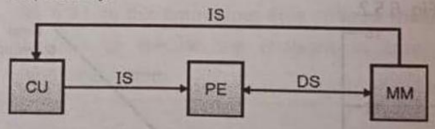


Fig. 6.7.1 : SISD computer

2. Single Instruction Multiple Data (SIMD) :

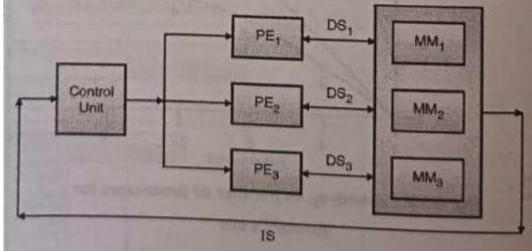
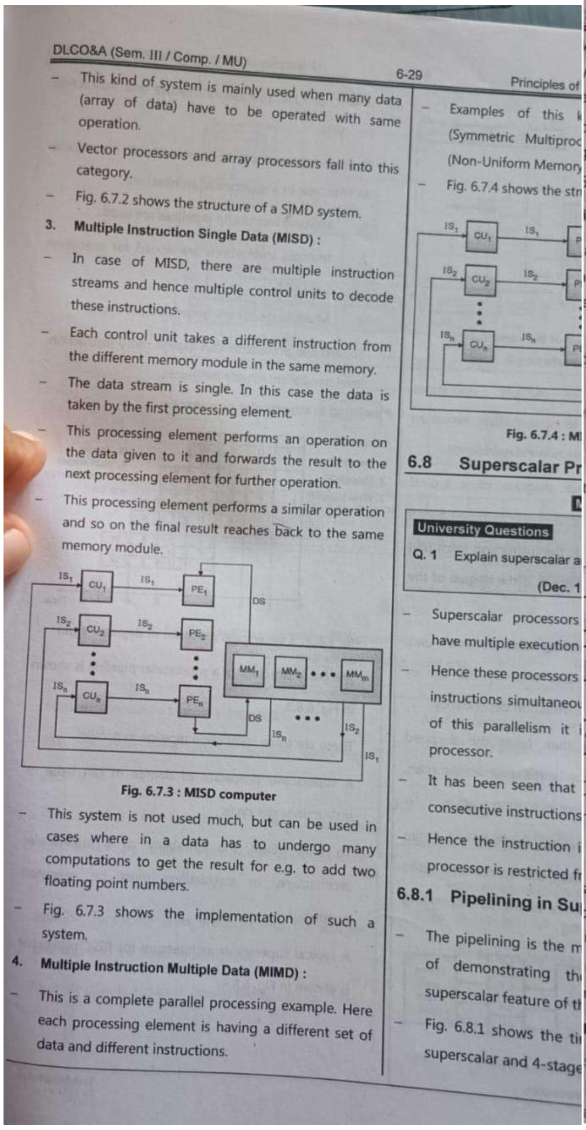


Fig. 6.7.2 : SIMD organization

- In this case the same instruction is given to multiple processing elements, but different data.



- This kind of system is mainly used when many data (array of data) have to be operated with same operation.
- Vector processors and array processors fall into this category.
- Fig. 6.7.2 shows the structure of a SIMD system.

3. **Multiple Instruction Single Data (MISD) :**

- In case of MISD, there are multiple instruction streams and hence multiple control units to decode these instructions.
- Each control unit takes a different instruction from the different memory module in the same memory.
- The data stream is single. In this case the data is taken by the first processing element.
- This processing element performs an operation on the data given to it and forwards the result to the next processing element for further operation.
- This processing element performs a similar operation and so on the final result reaches back to the same memory module.

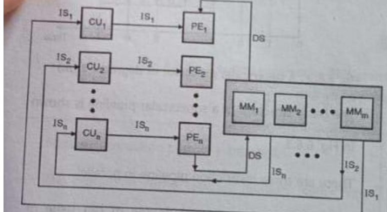
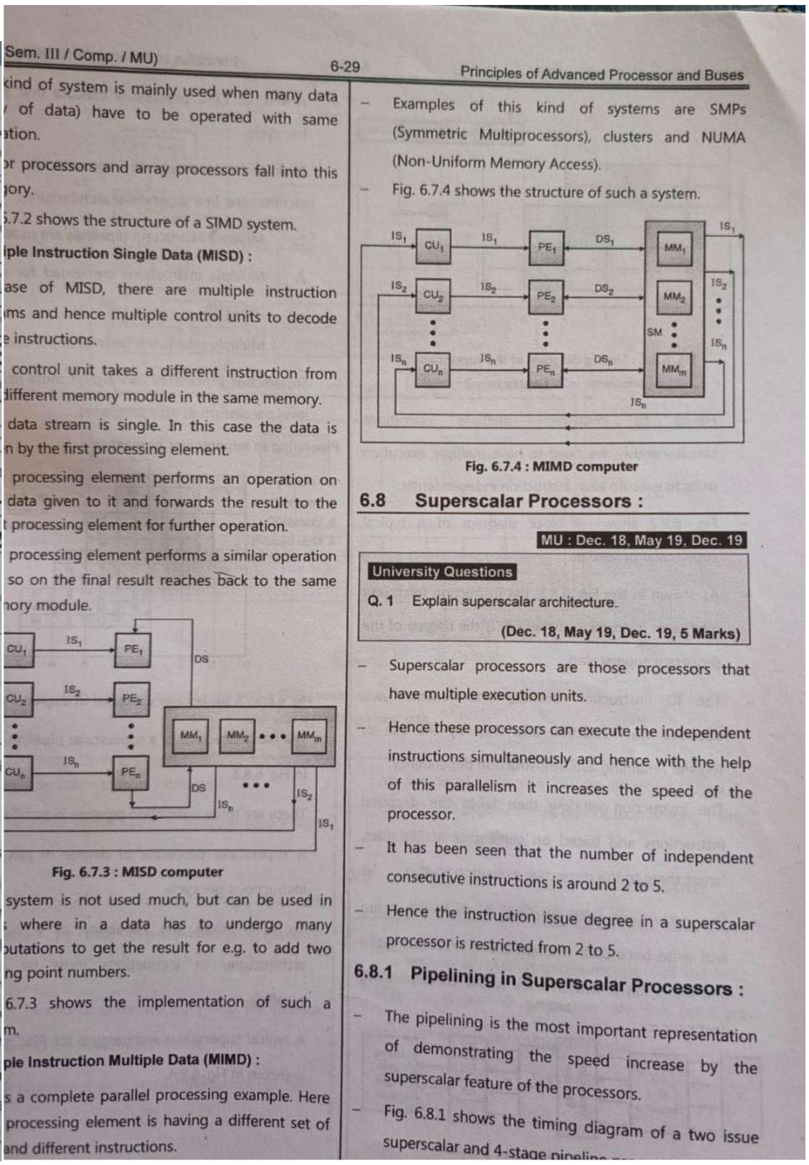


Fig. 6.7.3 : MISD computer

- This system is not used much, but can be used in cases where in a data has to undergo many computations to get the result for e.g. to add two floating point numbers.
- Fig. 6.7.3 shows the implementation of such a system.

4. **Multiple Instruction Multiple Data (MIMD) :**

- This is a complete parallel processing example. Here each processing element is having a different set of data and different instructions.



- Examples of this kind of systems are SMPs (Symmetric Multiprocessors), clusters and NUMA (Non-Uniform Memory Access).
- Fig. 6.7.4 shows the structure of such a system.

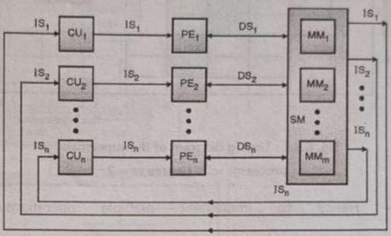


Fig. 6.7.4 : MIMD computer

6.8 Superscalar Processors :

MU : Dec. 18, May 19, Dec. 19

University Questions

Q. 1 Explain superscalar architecture.

(Dec. 18, May 19, Dec. 19, 5 Marks)

- Superscalar processors are those processors that have multiple execution units.
- Hence these processors can execute the independent instructions simultaneously and hence with the help of this parallelism it increases the speed of the processor.
- It has been seen that the number of independent consecutive instructions is around 2 to 5.
- Hence the instruction issue degree in a superscalar processor is restricted from 2 to 5.

6.8.1 Pipelining in Superscalar Processors :

- The pipelining is the most important representation of demonstrating the speed increase by the superscalar feature of the processors.
- Fig. 6.8.1 shows the timing diagram of a two issue superscalar and 4-stage

Q.9] Explain D, T, S-R and J-K flip-flops with truth tables.

Short Note on Flip-Flops. **05**

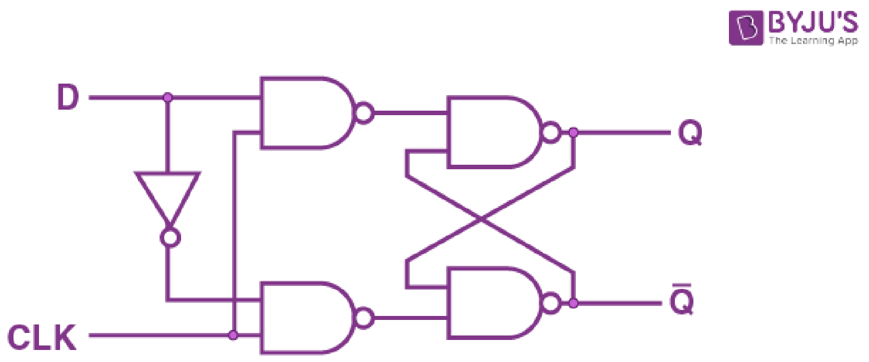
Ans:-

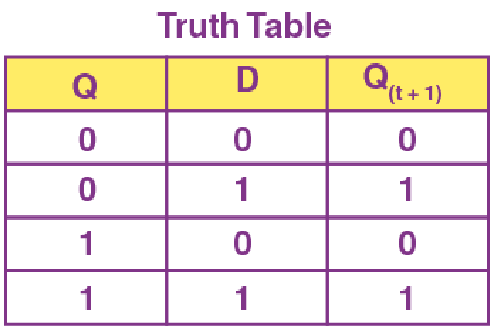
Part A:-

1. D Flip-Flop (Data / Delay Flip-Flop)

1. A D flip-flop is a memory device that stores **one bit** of data.
2. It has **one input, D**, and an output **Q**.
3. Whatever value is present on **D** at the moment of the clock pulse becomes the new output.
4. If **D = 0**, the output becomes **0** after the clock.
5. If **D = 1**, the output becomes **1** after the clock.
6. It avoids confusion because the output always follows the input.
7. It is commonly used in **registers, memory units, and data storage**.

8. It is often called a **delay flip-flop** because the input is transferred to the output with a delay of one clock.





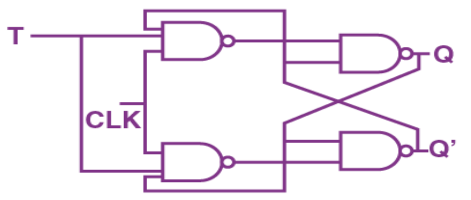
Truth Table

Q	D	$Q_{(t+1)}$
0	0	0
0	1	1
1	0	0
1	1	1

---

2. T Flip-Flop (Toggle Flip-Flop)

1. A T flip-flop is used when we want the output to **toggle**.
2. It has one input **T** and an output **Q**.
3. When **T = 0**, the output does **not change**.
4. When **T = 1**, the output **toggles** (switches from 0 to 1 or 1 to 0).
5. It is useful in **counters** because each clock pulse flips the output.
6. It is often used in **frequency division** (divide-by-2 circuits).
7. It can be made from a JK flip-flop by tying J = K = 1.
8. It is simple and widely used in sequential circuits.

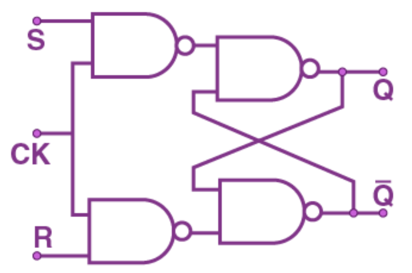


Truth Table

T	Q_N	Q_{N+1}
0	0	0
0	1	1
1	0	1
1	1	0

3. SR Flip-Flop (Set–Reset Flip-Flop)

1. SR flip-flop has two inputs: **S (Set)** and **R (Reset)**.
2. It stores **one bit** of information.
3. When **S = 1**, the output **Q becomes 1** (set state).
4. When **R = 1**, the output **Q becomes 0** (reset state).
5. When **S = 0 and R = 0**, the output does **not change**.
6. When **S = 1 and R = 1**, the output is **invalid**.
7. It is used in simple memory storage circuits.

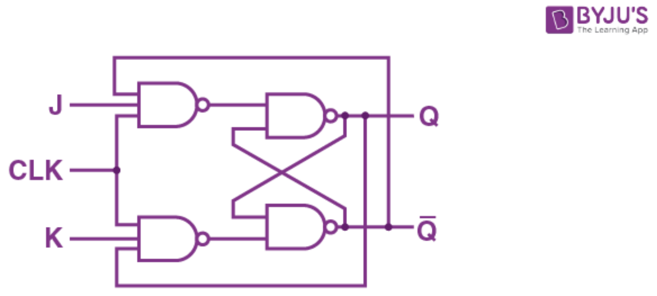


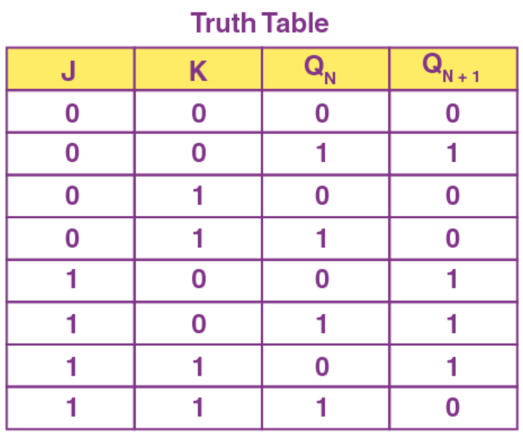
Truth Table

S	R	Q_N	Q_{N+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	-
1	1	1	-

4. JK Flip-Flop

1. JK flip-flop is an improved version of the SR flip-flop.
2. It has two inputs: **J** and **K**.
3. It removes the invalid condition present in the SR flip-flop.
4. When **J = 1 and K = 0**, the output becomes **1** (set).
5. When **J = 0 and K = 1**, the output becomes **0** (reset).
6. When **J = 0 and K = 0**, the output does **not change**.
7. When **J = 1 and K = 1**, the output **toggles**.
8. Used in counters, shift registers, and memory systems.




Truth Table

J	K	Q_N	Q_{N+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Part B:- Short Note on Flip-Flops

Flip-flops are basic **sequential logic circuits** used to store **one bit** of data. They are the fundamental building blocks of memory units, counters, registers, and many digital systems. Flip-flops change their output only when a clock signal is applied, which is why they are also called **bistable multivibrators** (having two stable states: 0 and 1).

There are four commonly used flip-flops:

---

1. SR Flip-Flop (Set–Reset)

The SR flip-flop has two inputs: **S (set)** and **R (reset)**.
When S = 1, the output becomes 1; when R = 1, the output becomes 0.
When both inputs are 0, the output remains unchanged.
The input combination S = 1 and R = 1 is considered invalid.

---

2. JK Flip-Flop

The JK flip-flop is an improved version of the SR flip-flop, removing the invalid condition.
Inputs are **J** and **K**.
When J = 1 and K = 0, the output is set; when J = 0 and K = 1, it resets.
When both J and K are 1, the output **toggles** from 0 to 1 or 1 to 0.

---

3. D Flip-Flop (Data/Delay)

The D flip-flop has a single input **D**.
The output simply follows the input on the next clock pulse.
If D = 0, output becomes 0; if D = 1, output becomes 1.
It is widely used in registers and memory because it gives clean, predictable output.

---

4. T Flip-Flop (Toggle)

The T flip-flop toggles its output when T = 1.
If T = 0, the output stays the same.
This flip-flop is used in **counters** and **frequency division** circuits.
It changes state (0 → 1 or 1 → 0) on every clock when T = 1.

Q.10] Explain Register Organization of a processor. **05**

Ans:-

Register Organization of a Processor

Register organization refers to a set of small, high-speed storage locations inside the CPU that hold data, instructions, addresses, and intermediate results while a program is running. These registers are the fastest memory in the computer and help the processor work quickly without depending on slow main memory. The organization explains how all these registers are arranged, how they communicate with the ALU, and how they support the execution of instructions.

Register Organization of a Processor

The register organization of a processor consists of **two main types** of registers:

1. **User-Visible Registers**
2. **Control and Status Registers**

---

1. User-Visible Registers

Definition:

These are registers that can be accessed directly by the programmer or by machine instructions to store data, addresses, and intermediate results.

Uses:

They help in arithmetic operations, memory addressing, procedure calls, and holding temporary values needed during instruction execution.

Examples:

Below are the sub-types you asked for.

(a) Data Registers

1. **Definition:** Hold data values and operands for arithmetic and logic operations.
2. **Uses:** Used during addition, subtraction, multiplication, comparisons, and temporary storage.
3. **Example:** AX, BX in x86 processors.

(b) Address Registers

1. **Definition:** Store memory addresses used to access data or instructions.
2. **Uses:** Used for pointing to memory locations, arrays, tables, and indirect addressing.
3. **Example:** Index register, base register.

(c) Floating-Point Registers

1. **Definition:** Hold decimal/real numbers used for floating-point arithmetic.
2. **Uses:** Used for scientific, graphics, and mathematical calculations.
3. **Example:** F0–F7 in many architectures.

(d) Stack Pointer (SP)

1. **Definition:** Stores the address of the top of the stack in memory.
2. **Uses:** Used for function calls, returns, parameter passing, and push/pop operations.
3. **Example:** SP register in x86.

---

2. Control and Status Registers

Definition:

These registers control the operation of the CPU and indicate the status of the current instruction and results.

Uses:

They help in sequencing instructions, monitoring CPU conditions, handling memory access, and managing execution flow.

Examples:

Below are the sub-types you listed.

(a) Program Counter (PC)

1. **Definition:** Stores the address of the next instruction to be executed.
2. **Uses:** Controls the flow of the program by deciding the next instruction.
3. **Example:** PC in all CPUs.

(b) Instruction Register (IR)

1. **Definition:** Holds the instruction that is currently being executed.
2. **Uses:** Used during instruction decoding and control signal generation.
3. **Example:** IR in all processors.

(c) Memory Address Register (MAR)

1. **Definition:** Stores the address of the memory location that the CPU wants to access.
2. **Uses:** Used during fetch and memory read/write operations.
3. **Example:** MAR in Von Neumann model.

(d) Memory Buffer Register / Memory Data Register (MBR/MDR)

1. **Definition:** Temporarily holds data moving between CPU and memory.
2. **Uses:** Used when reading data from memory or writing data to memory.
3. **Example:** MBR/MDR in typical CPU structures.

(e) Status Register / Condition Code Register (CCR)

1. **Definition:** Contains flags that indicate results of operations.
2. **Uses:** Helps in decision-making operations like branching, comparisons, and arithmetic checks.
3. **Example:** Zero flag, Carry flag, Sign flag, Overflow flag.

Q.11] Describe Half Adder and Full Adder circuit with block and logical diagram & truth table.

Design full adder using Half Adder. **05**

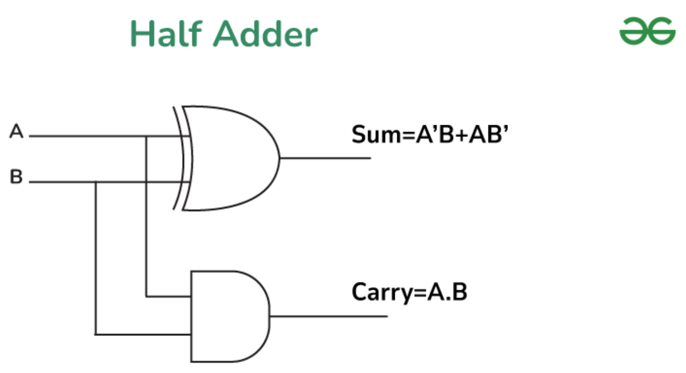
Ans:- PartA:-

What is Half Adder?

Half adder is a combinational circuit that is used to add two 1-bit inputs to generate two outputs sum and carry. The sum in half adder is given by XORing both the inputs. The carry in the half adder is given by the product of both inputs. Half Adders are used in the Various Digital Systems Where Addition of Binary Numbers is Required Such as Arithmetic Circuits, Digital Calculators, Microcontrollers and Processors, Communication systems and Control Systems.

Logic Diagram for Half Adder

To implement half adder, we require one XOR gate and one AND gate. Below is the logic diagram for half adder.

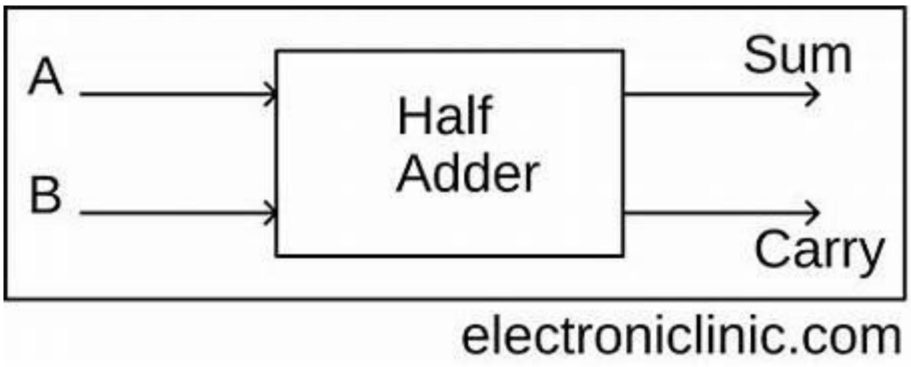


Truth Table

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Block Diagram for Half Adder

Below is the block diagram for half adder.



PartB:-

Full Adder is a combinational circuit that adds three inputs and produces two outputs. The first two inputs are A and B and the third input is an input carry as C-IN. The output carry is designated as C-OUT and the normal output is designated as S which is SUM.

- The C-OUT is also known as the majority 1's detector, whose output goes high when more than one input is high.
- A full adder logic is designed in such a manner that can take eight inputs together to create a byte-wide adder and cascade the carry bit from one adder to another.
- We use a full adder because when a carry-in bit is available, another 1-bit adder must be used since a 1-bit half-adder does not take a carry-in bit.
- A 1-bit full adder adds three operands and generates 2-bit results.

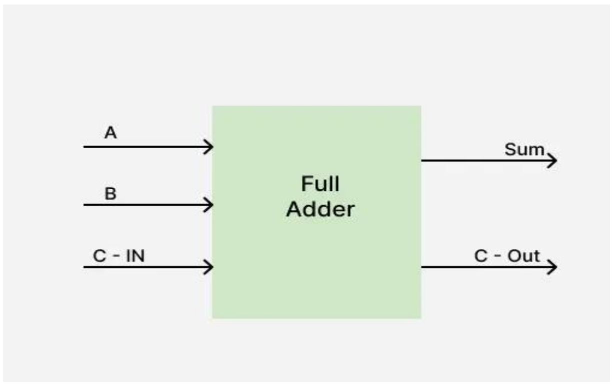
Full Adder Truth Table

A Full Adder takes three binary inputs:

- A (first bit)
- B (second bit)
- C-IN (carry input)

And it produces two outputs:

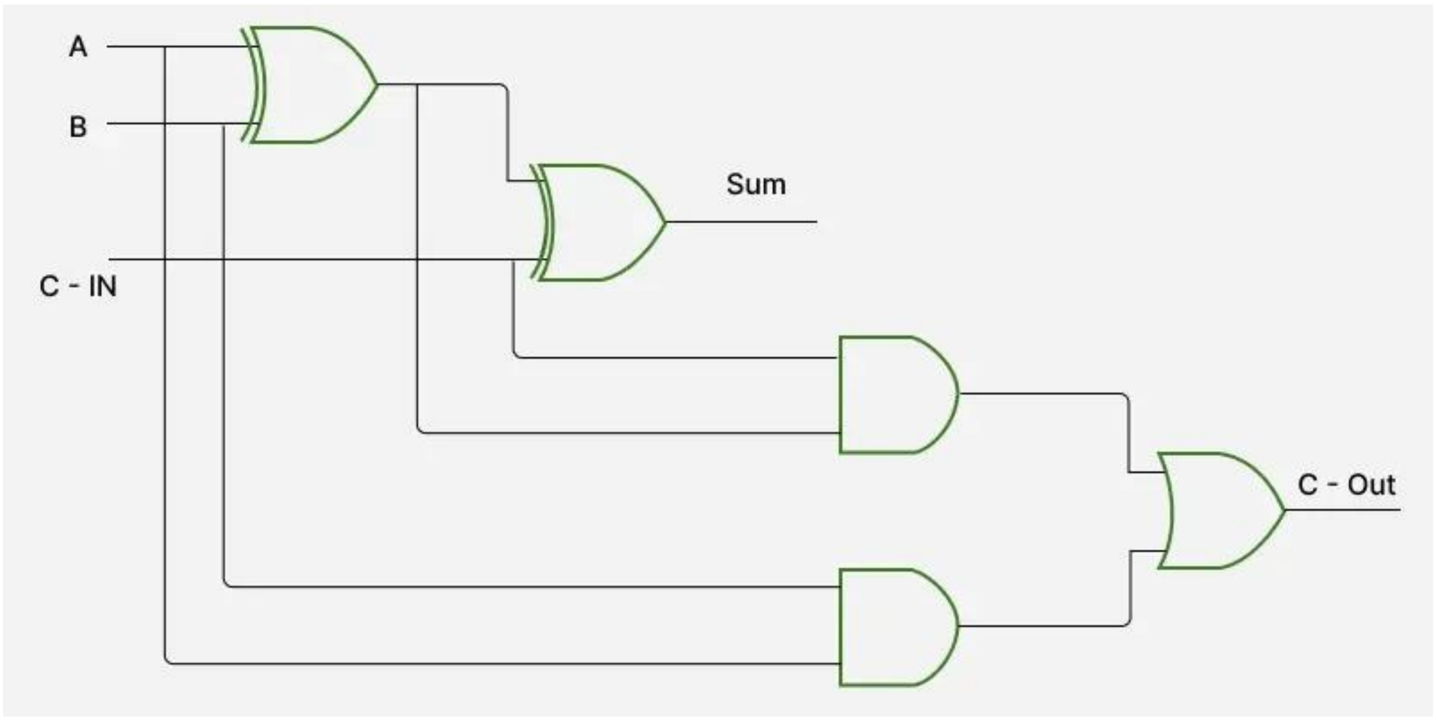
- Sum (S)
- Carry Out (C-OUT)



BLOCK DAIGRAM

Here’s the truth table for the full adder and Logical Diagram:

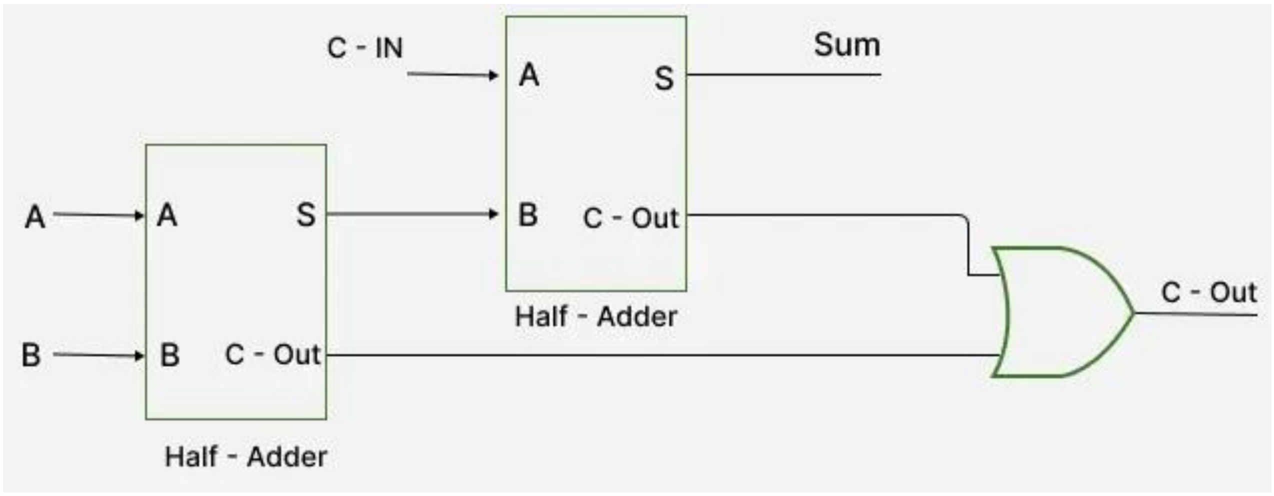
INPUT			**OUTPUT**	
A	**B**	**C-IN**	**Sum**	**C-OUT**
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



PART C:-

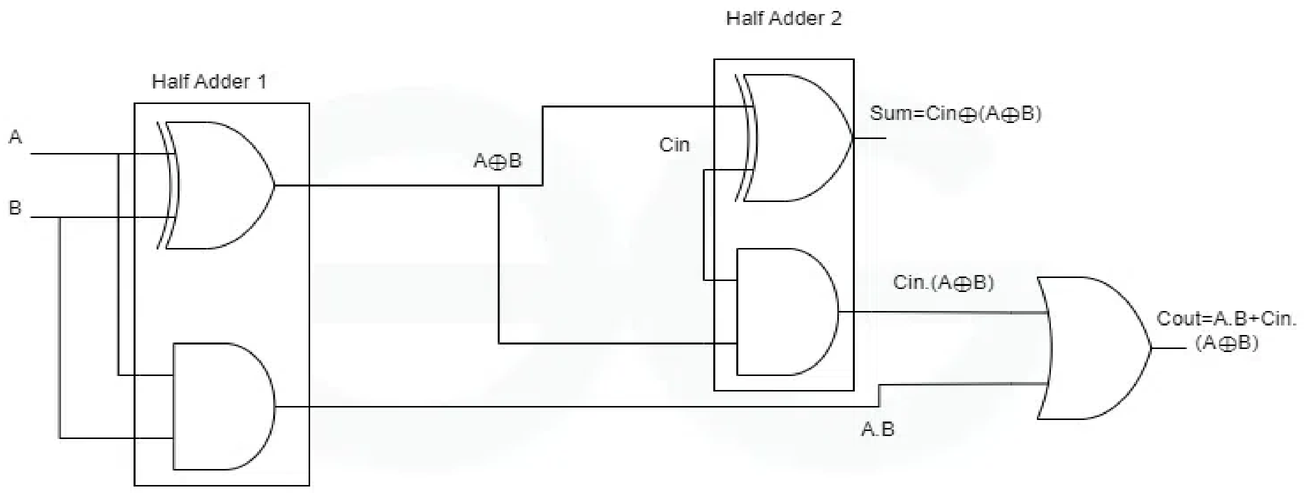
Implementation of Full Adder using Half Adders

2 Half Adders and an OR gate is required to implement a Full Adder.



Full Adder using Half Adders

With this logic circuit, two bits can be added together, taking a carry from the next lower order of magnitude, and sending a carry to the next higher order of magnitude.



Implementation of Full Adder Using Half Adders

Application of Full Adder in Digital Logic

- **Arithmetic circuits:** Full adders are utilized in math circuits to add twofold numbers. At the point when different full adders are associated in a chain, they can add multi-bit paired numbers.
- **Data handling:** Full adders are utilized in information handling applications like advanced signal handling, information encryption, and mistake rectification.
- **Counters:** Full adders are utilized in counters to addition or decrement the count by one.
- **Multiplexers and demultiplexers:** Full adders are utilized in multiplexers and demultiplexers to choose and course information.
- **Memory tending to:** Full adders are utilized in memory addressing circuits to produce the location of a particular memory area.
- **ALUs:** Full adders are a fundamental part of Number juggling Rationale Units (ALUs) utilized in chip and computerized signal processors.

Q.12] Explain IEEE-754 floating point representation. **05**

Ans:-

IEEE-754 Floating Point Representation

IEEE-754 is the standard method used by computers to store and represent real numbers (decimal numbers, fractions, very small and very large values). It allows the computer to store numbers in a scientific notation format, just like we write numbers as:

± M × 2^E
Where:

- **Sign bit** shows whether the number is positive or negative
- **M (Mantissa or significand)** stores the value of the number

- **E (Exponent)** controls how large or small the number is

The IEEE-754 format ensures the same representation across all computers so that results are consistent everywhere.

Two Common IEEE-754 Formats

1. Single Precision (32-bit)

It uses **32 bits** divided as:

Part	**Bits**
Sign	1 bit
Exponent	8 bits
Mantissa (Fraction)	23 bits

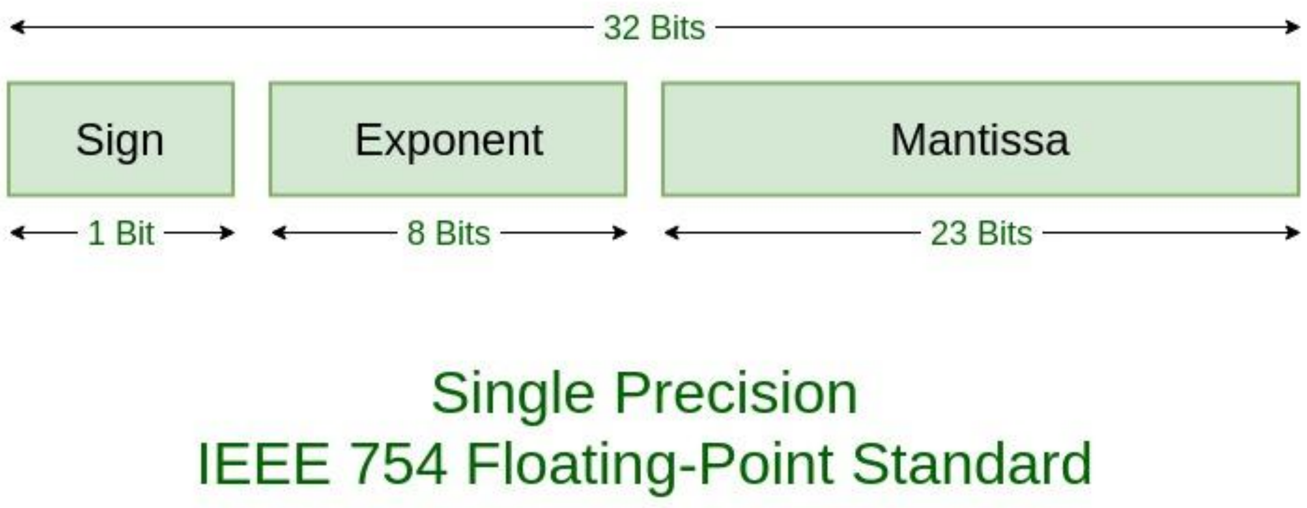
---

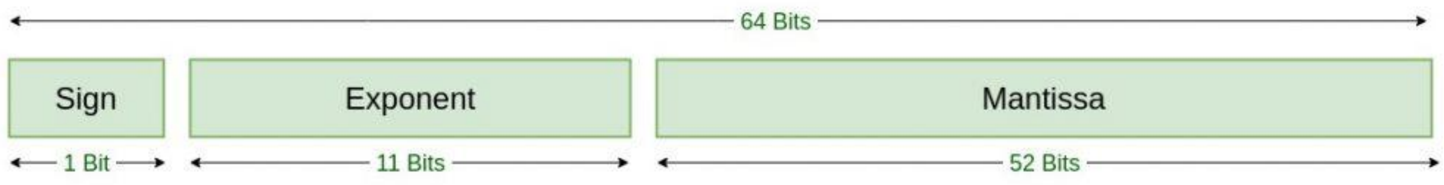
2. Double Precision (64-bit)

It uses **64 bits** divided as:

Part	**Bits**
Sign	1 bit
Exponent	11 bits
Mantissa (Fraction)	52 bits

General Format of IEEE-754





Double Precision
IEEE 754 Floating-Point Standard

Explanation of Each Field

1. Sign Bit (S)

- **0 → Positive number**
- **1 → Negative number**

---

2. Exponent (E)

- Stores the exponent using **biased notation**.
- A bias is added to avoid negative exponent values.

Bias values:

- For **32-bit:** bias = **127**
- For **64-bit:** bias = **1023**

Actual exponent = Stored exponent – Bias

---

3. Mantissa / Fraction (M)

- Stores the fractional part of the number.
- The IEEE-754 format assumes a **hidden leading 1**, except for special cases.
  Example: the number is stored as **1.fraction**

How a Number Is Represented

IEEE-754 stores a number as:

Value = (–1)^S × 1.Mantissa × 2^(Exponent – Bias)

This gives a wide range of decimal values, including very small numbers and very large numbers.

Example Overview

To represent **–5.75** in 32-bit format:

1. Convert to binary → **101.11**
2. Normalize → **1.0111×2^2**
3. Sign = 1 (negative)
4. Exponent = 2 + 127 = **129** → binary **10000001**
5. Mantissa = **011100000....**

Special Values in IEEE-754

Condition	**Representation**
Zero	all exponent bits = 0, mantissa = 0
Infinity	exponent all 1s, mantissa = 0
NaN (Not a Number)	exponent all 1s, mantissa ≠ 0
Denormal numbers	exponent all 0s, used for very small values

Advantages of IEEE-754

1. Provides a standard representation for real numbers on all computers.
2. Supports very large and very small numbers.
3. Offers high precision using mantissa bits.
4. Supports special values like infinity and NaN.
5. Reduces rounding errors using specific rounding rules.

Q.13] Explain the instruction cycle with state diagram. **10**

Ans:-

Explain the Instruction Cycle with State Diagram

The instruction cycle is the complete process that the CPU follows to execute a single instruction. Every instruction in a program goes through a series of basic steps inside the processor. These steps make sure the CPU fetches the instruction, understands it, gets the required data, performs the operation, and then stores the result. This cycle repeats continuously until the program ends.

The instruction cycle ensures smooth communication between the CPU, memory, and input/output devices. It is controlled by the control unit and uses important registers like the Program Counter (PC), Instruction Register (IR), Memory Address Register (MAR), and Memory Data Register (MDR).

Main Phases of the Instruction Cycle

1. Fetch Cycle

In this stage, the processor brings the instruction from memory.

Steps:

- The **Program Counter (PC)** gives the address of the next instruction.
- This address is moved to the **Memory Address Register (MAR)**.
- The instruction is fetched from memory into the **Memory Data Register (MDR)**.
- The fetched instruction is placed into the **Instruction Register (IR)**.
- The PC is increased to point to the next instruction.

This completes the fetch phase.

---

2. Decode Cycle

The CPU now interprets what the instruction means.

Steps:

- The control unit reads the contents of the IR.
- It identifies the operation type (add, move, compare, jump, etc.).
- It identifies which registers, data, and addressing modes are required.
- The control unit prepares control signals based on the instruction.

---

3. Operand Fetch (Optional)

If the instruction needs data (operands), the CPU fetches it from memory or registers.

Steps:

- If data is in a register, it is read directly.
- If data is in memory, the address is loaded into MAR and the value is fetched into MDR.
- Operands are then supplied to the ALU.

This step may be skipped for simple instructions that operate directly on registers.

---

4. Execute Cycle

The ALU or control unit performs the actual operation of the instruction.

Examples of operations:

- Arithmetic operations (add, subtract, multiply)
- Logical operations (AND, OR, NOT)
- Data transfer (move, load, store)
- Branching or jumping to another location
- Comparison operations

After the operation is performed, results are produced.

---

5. Write-Back Cycle

The final result produced by the execution step is saved.

Steps:

- If the result belongs to a register, it is written back to that register.
- If the result belongs to memory, the address is placed in MAR and data in MDR and then written to memory.

This completes the entire instruction cycle.

---

6. Interrupt Cycle (If Any)

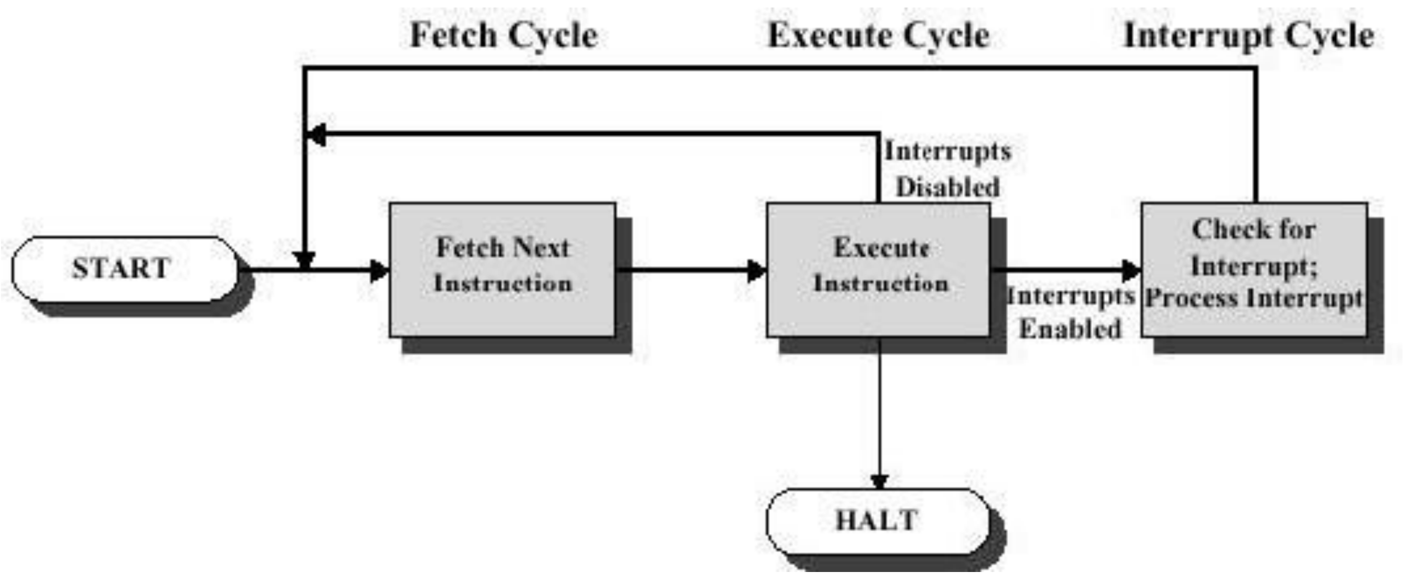
If an interrupt occurs, the processor temporarily stops the current instruction sequence.

Steps:

- The CPU saves the current PC value.
- It jumps to the interrupt service routine.
- After handling the interrupt, the CPU returns to the saved PC and continues.

Interrupt cycle does not occur every time—it only occurs when an interrupt signal is present.

State Diagram of the



The state diagram shows the flow from one phase to another.
The usual order is:

Fetch → Decode → Operand Fetch → Execute → Write Back → Fetch (next instruction)

If an interrupt occurs, the sequence temporarily breaks:

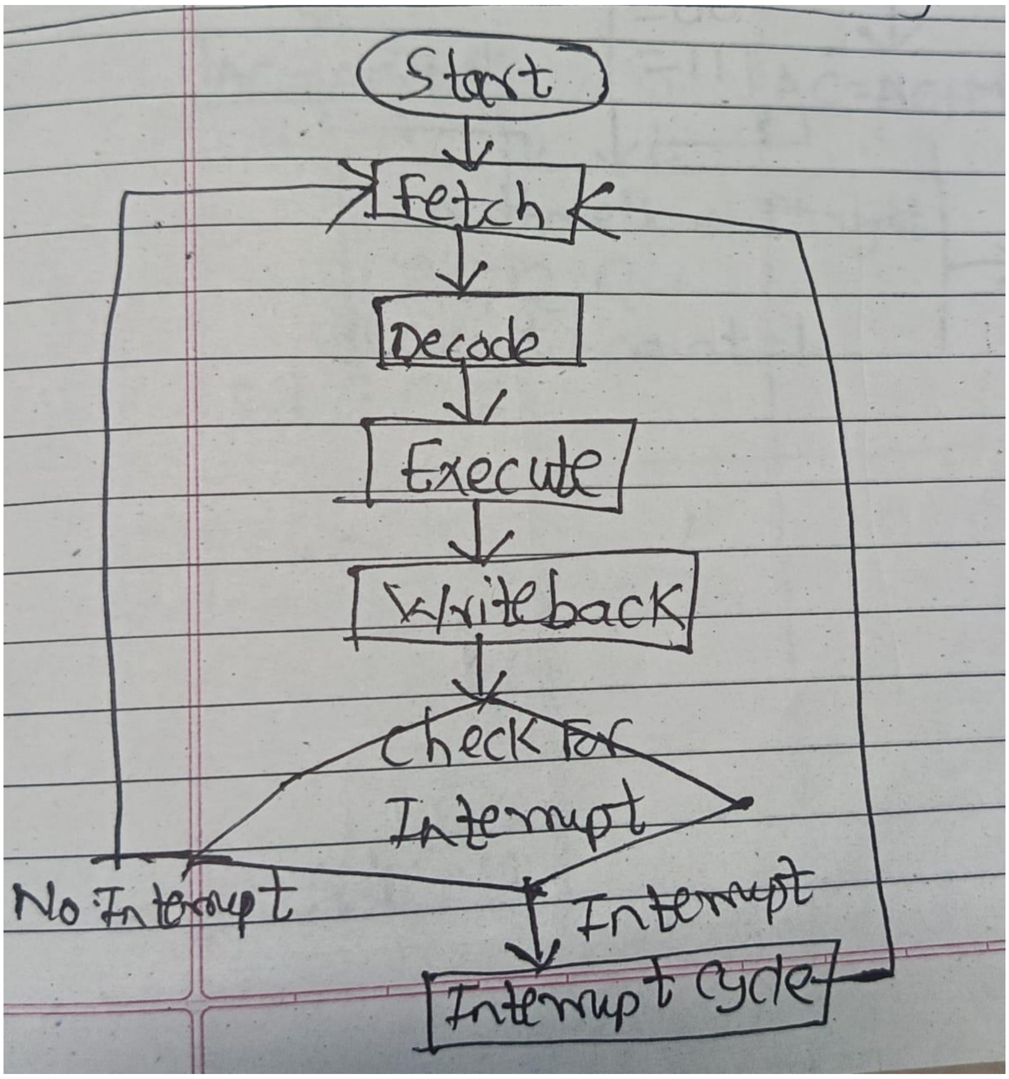
Execute → Interrupt → Fetch

This diagram represents how the CPU moves between different states while completing each instruction.

Detailed Explanation of the State Diagram

1. **Fetch State:** CPU reads the instruction from memory.
2. **Decode State:** CPU breaks down the instruction and prepares control signals.
3. **Operand Fetch State:** CPU fetches data needed for execution.
4. **Execute State:** CPU performs arithmetic, logical, or control operations.
5. **Write-Back State:** CPU writes results to registers or memory.
6. **Interrupt State:** CPU checks and handles interrupt requests before continuing the next instruction.

Each state transitions smoothly to the next, creating a continuous cycle of instruction execution.



a). Cycle begins at Start state, which transitions to Fetch.

b] After Writeback, CPU checks if any interrupts have occurred.

c] If no interrupt is present cycle returns to Fetch to get next instruction.

d). If an interrupt is pending, CPU performs Interrupt Cycle before returning to Fetch for next instruction

Q.14] Write a short note on Encoder and Decoder. **05**

Ans:- PartA:-

Encoder (Easy and Detailed Answer)

An **Encoder** is a digital circuit that converts **2^n input lines into n output lines**. It takes multiple input signals, but only **one input is active at a time**, and it produces a binary code corresponding to that active input. In simple words, an encoder reduces many inputs into a smaller number of outputs by generating a binary representation.

Encoders are used in digital systems to save hardware, reduce wiring, and convert physical positions or signals into binary codes that a computer can understand. They are commonly used in keyboards, communication systems, sensors, and control circuits.

Working Principle (Simple Words)

- The encoder checks which input line is HIGH (1).
- It then outputs the **binary code** for that input position.
- For example, if input line 5 is active, the encoder outputs the binary number **101**.
- Only **one input** must be active; otherwise, the encoder will not work correctly.

Types of Encoders

1. 4-to-2 Encoder

- Has 4 input lines and 2 output lines.
- Converts one active input into a 2-bit binary code.

Example:

- Input 00 → Output 00
- Input 01 → Output 01
- Input 10 → Output 10
- Input 11 → Output 11

---

2. 8-to-3 Encoder

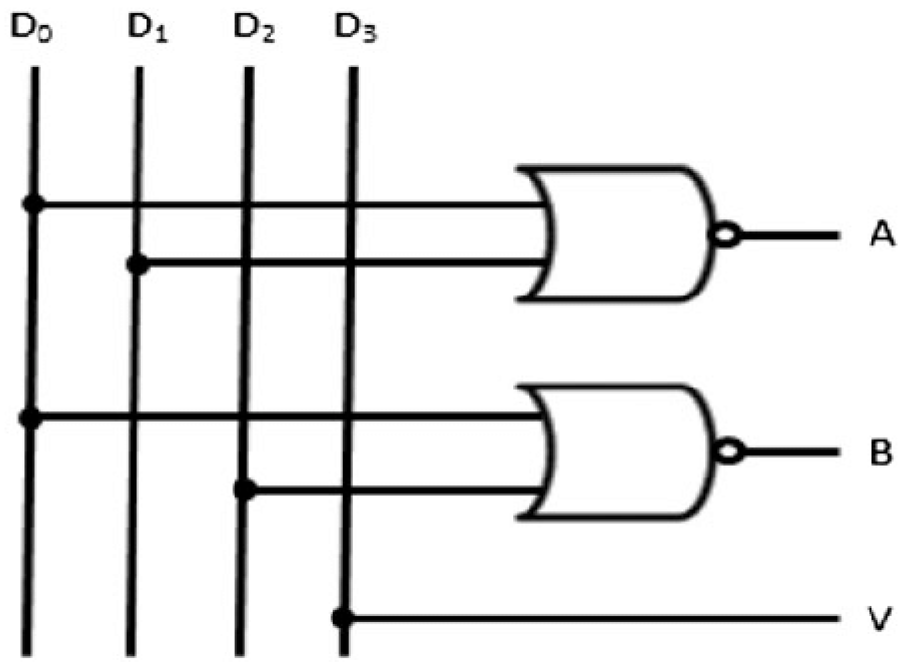
- Has 8 input lines and 3 output lines.
- Converts one active line out of 8 into a 3-bit binary number.

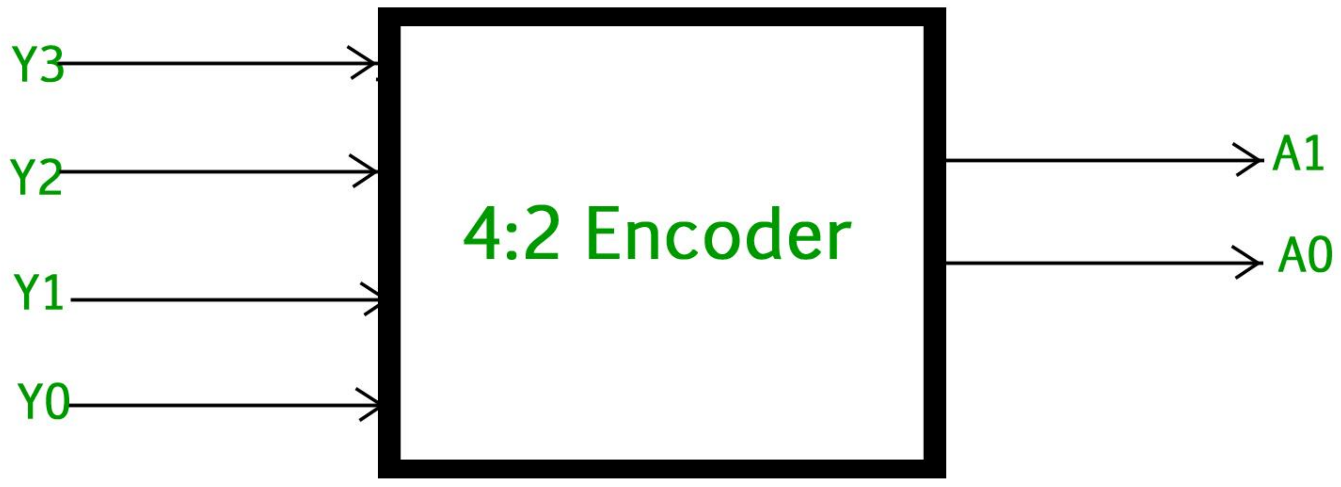
---

3. Priority Encoder

A special type of encoder where inputs have priority.
If more than one input is active, the encoder outputs the code of the **highest-priority** input.
This solves the problem of multiple active inputs.

Block Diagram of Encoder





Truth Table Example (4-to-2 Encoder)

Input D3 D2 D1 D0 Output Y1 Y0

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

- D3 is highest priority
- Output (Y1 Y0) gives the binary value of the active input

Applications of Encoders

1. **Keyboards:** Convert key presses into binary codes for the CPU.
2. **Robotics:** Convert motion or position into digital signals.
3. **Communication systems:** Encode signals before transmission.
4. **Control systems:** Convert sensor information into binary form.
5. **Memory addressing:** Selects which memory block or device to use.

Advantages

1. Saves wiring and reduces circuit complexity.
2. Converts many inputs into few outputs efficiently.
3. Useful in simplifying digital designs.
4. Easy to integrate with microprocessors.
5. Works well in communication and control systems.

Conclusion

An encoder is a useful digital device that converts one active input into a binary code. It minimizes hardware, speeds up processing, and is widely used in digital electronics, communication, and computer systems. Priority encoders improve accuracy when multiple inputs occur at the same time.

Part B:-

Decoder (Easy and Detailed Answer)

A **Decoder** is a digital circuit that converts **n input lines into 2^n output lines**. It performs the opposite function of an encoder. A decoder takes a binary input code and activates **exactly one output** based on that code. In simple words, a decoder **detects** which binary number is given as input and turns ON the corresponding output line.

Decoders are widely used in memory systems, communication systems, and digital devices where specific selection or identification is needed.

Working Principle (Simple Words)

- The decoder receives an **n-bit binary input**.
- It converts that input into **one active output** out of 2^n possible outputs.
- Only **one output line becomes 1**, and all others remain 0.
- For example, if the input is **10 (binary)**, the **third output** becomes active.

This is why decoders are also called **binary-to-decimal converters**.

Types of Decoders

1. 2-to-4 Decoder

- Has 2 input lines and 4 output lines.
- Converts a 2-bit binary number into one active output.
- If input = 01 → Output line 1 is HIGH.

---

2. 3-to-8 Decoder

- Has 3 input lines and 8 output lines.
- Converts a 3-bit input (000 to 111) into one of 8 outputs.

---

3. 4-to-16 Decoder

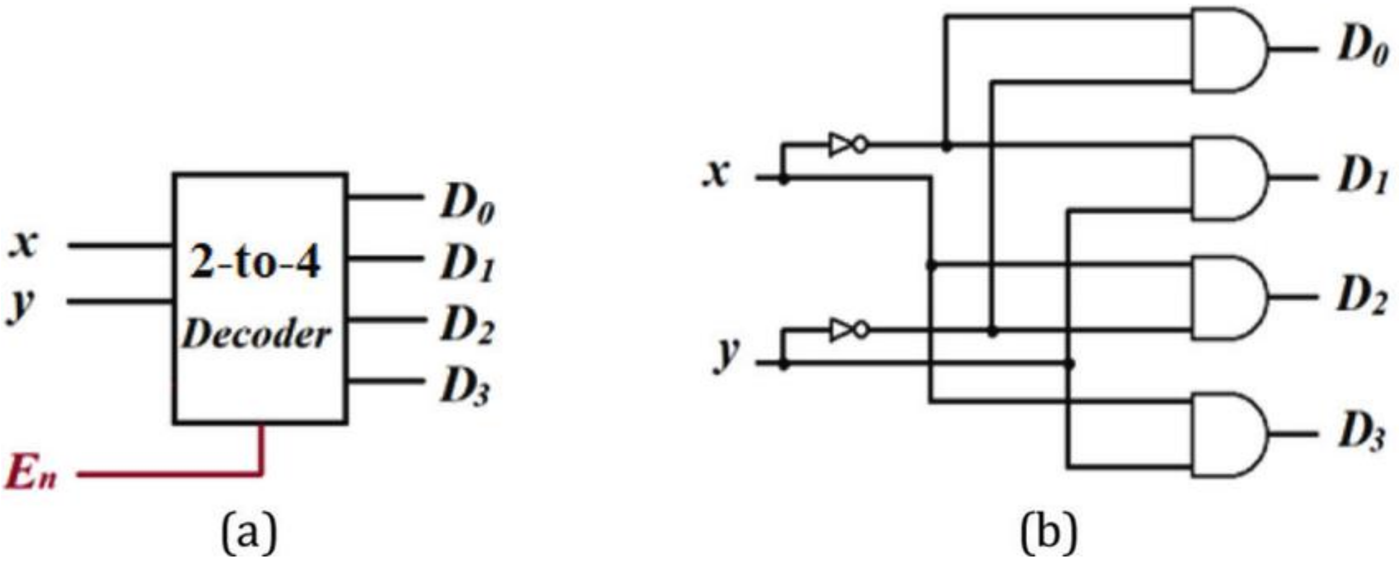
- Has 4 binary inputs and 16 outputs.
- Used in memory addressing and large digital systems.

---

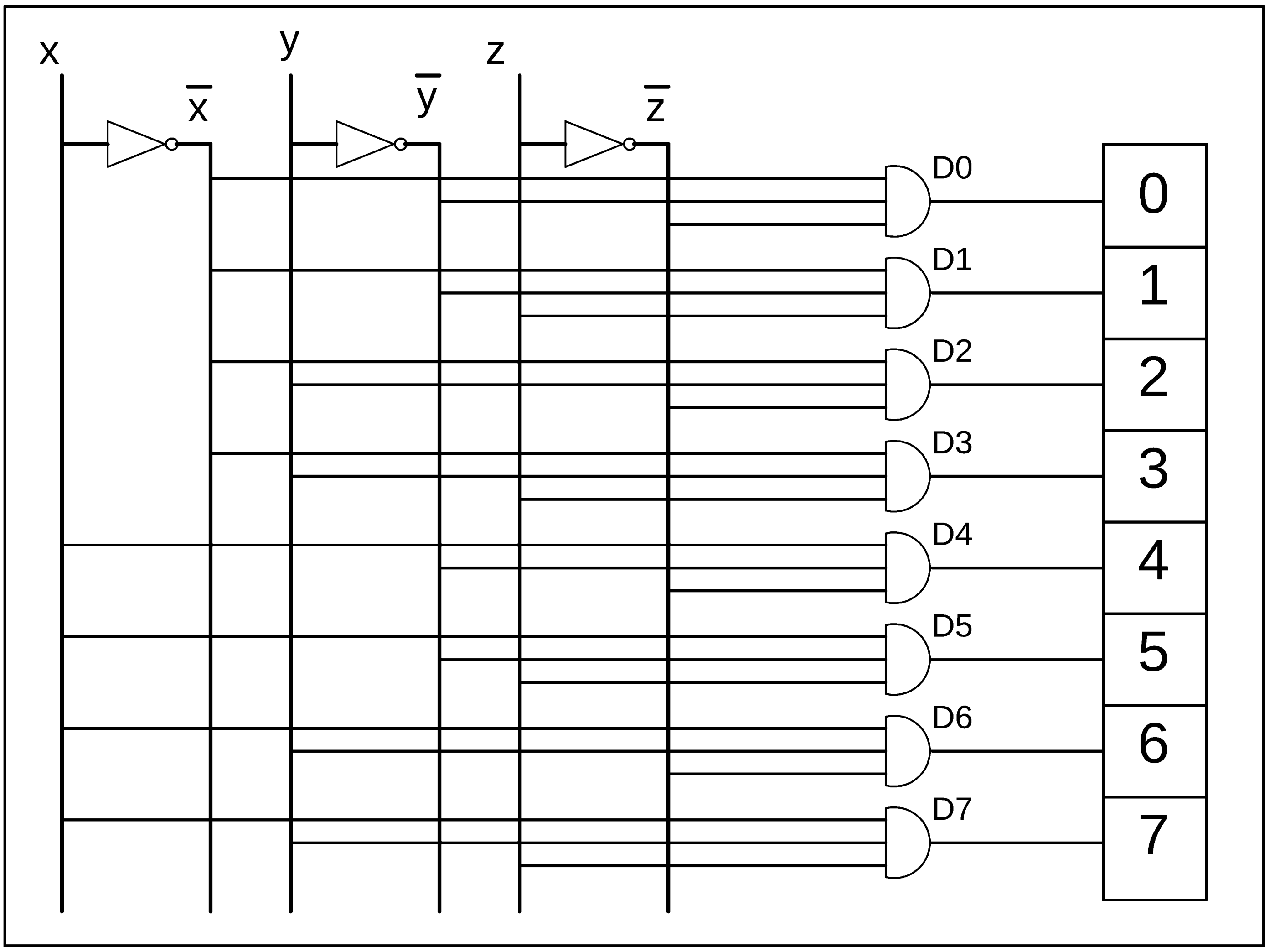
4. Decoder with Enable Input

- Has an extra **Enable (E)** signal.
- The decoder works only when E = 1.
- Very useful in chip selection and memory design.

Block Diagram of a Decoder



3 x 8 line Decoder Logic Diagram



electroniclinic.com

Truth Table Example (2-to-4 Decoder)

Enable	**A1**	**A0**	**Y3**	**Y2**	**Y1**	**Y0**
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

When **Enable = 0**, no output is activated (all 0).

Applications of Decoders

1. **Memory Address Decoding:** Used to select specific memory locations.
2. **Instruction Decoding:** Helps CPUs understand and decode machine instructions.
3. **Data Routing:** Sends data to the correct output device.
4. **Seven-Segment Displays:** Converts binary input to display digits.
5. **Chip Selection:** Used in microprocessors to enable specific chips.

Advantages

1. Converts binary inputs into unique outputs easily.
2. Useful for selecting memory locations or hardware devices.
3. Reduces complexity in digital systems.
4. Works as a binary-to-decimal converter.
5. Helps CPUs identify instructions quickly.

Conclusion

A decoder is a digital circuit that takes an n-bit binary input and activates one of the 2^n outputs. It is widely used in memory systems, displays, communication circuits, and microprocessors. Decoders play a key role in selecting, identifying, and routing information inside digital systems.

Q.15] Explain Master-Slave JK Flip-Flop with PRESET & CLEAR. **10**

Ans:-

Race-Around Condition in JK Flip-Flop
In a JK flip-flop, when **J = K = 1** and the **clock stays HIGH for a long time**, the output **Q keeps toggling**

again and again.
This makes the output **unstable and unpredictable**.
This problem is called the **Race-Around Condition**.
To avoid this, the clock must stay HIGH only for a very short time.
To solve this permanently, the **Master–Slave JK Flip-Flop** was introduced.

Master–Slave JK Flip-Flop
A Master–Slave JK flip-flop is made by **connecting two JK flip-flops in series**.
The first one is called the **Master**, and the second one is called the **Slave**.

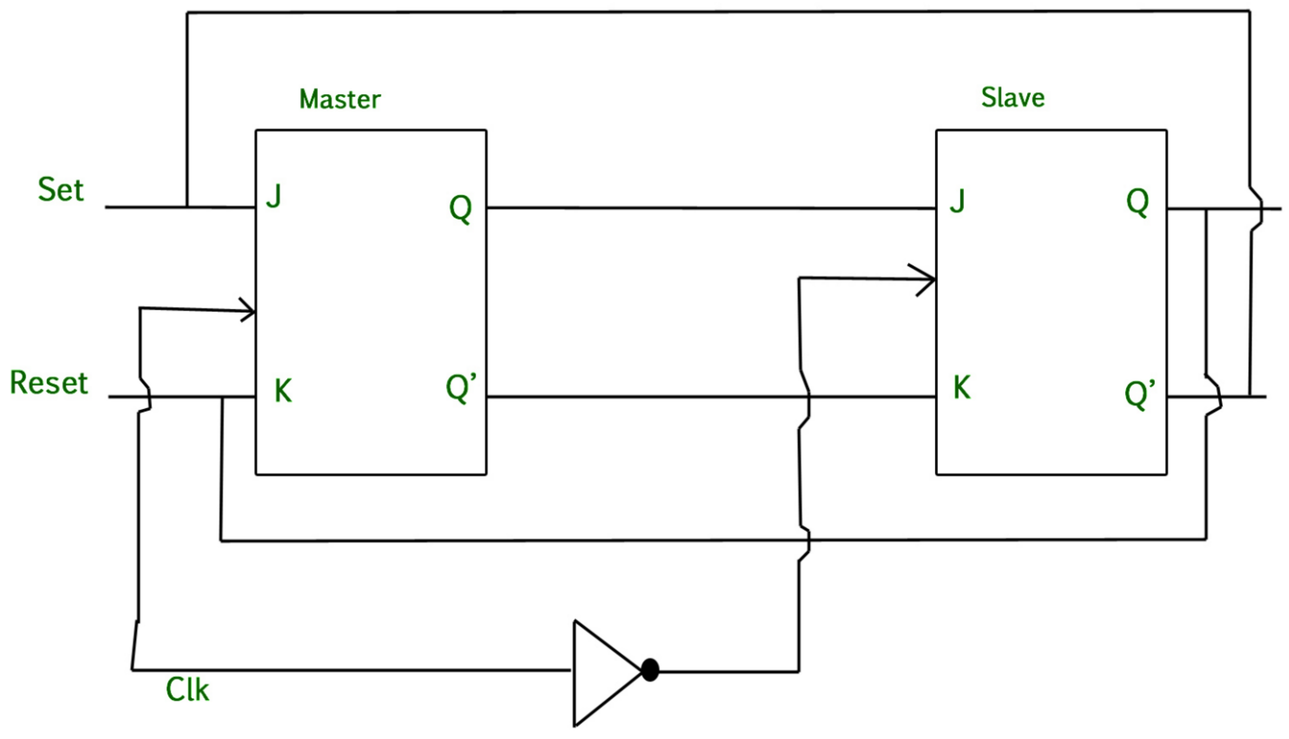
- The **Master** takes the input first.
- The **Slave** updates the final output later.

The Master's output goes into the Slave's inputs.
The Slave's output is also fed back to the Master, ensuring correct operation.

An **inverter** is used with the clock signal so that:

- When **clock = 1**, **Master works** and **Slave is OFF**
- When **clock = 0**, **Slave works** and **Master is OFF**

Because they work in **opposite clock phases**, the output changes **only once per clock cycle**, removing the race-around problem.



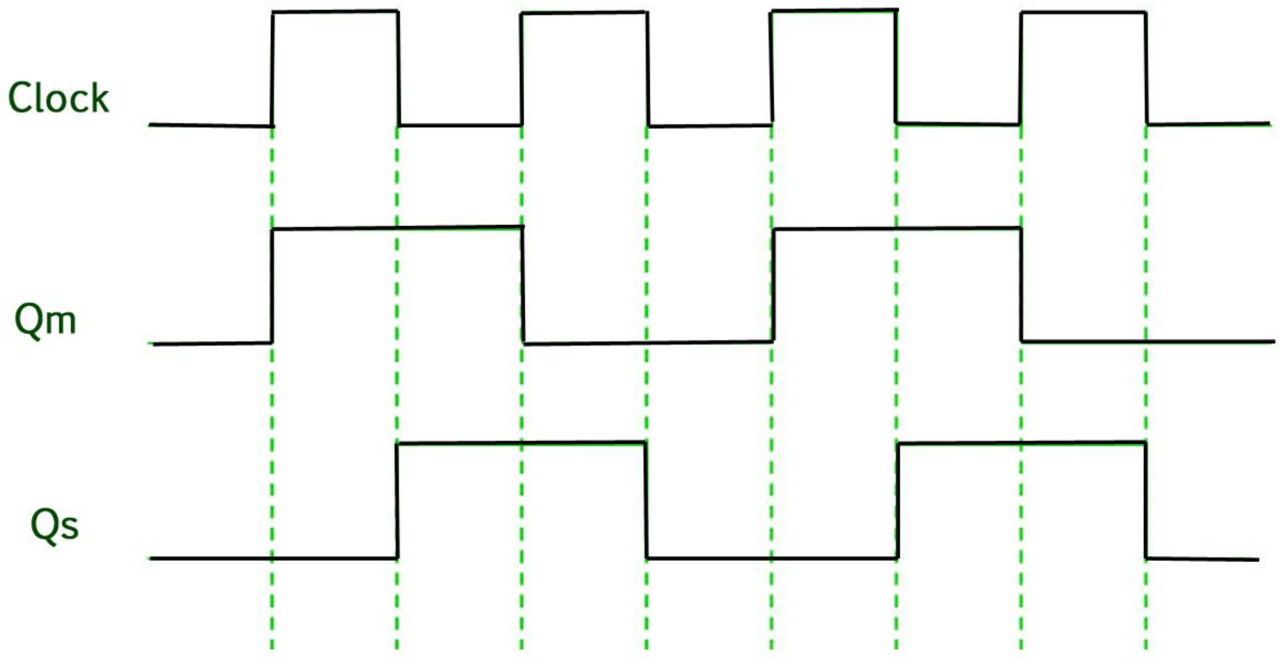
Master-slave JK Flip-Flop

Working of a master slave flip flop -

1. When the clock pulse goes to 1, the slave is isolated; J and K inputs may affect the state of the system. The slave flip-flop is isolated until the CP goes to 0. When the CP goes back to 0, information is passed from the master flip-flop to the slave and output is obtained.
2. Firstly the master flip flop is positive level triggered and the slave flip flop is negative level triggered, so the master responds before the slave.
3. If J=0 and K=1, the high Q' output of the master goes to the K input of the slave and the clock forces the slave to reset, thus the slave copies the master.
4. If J=1 and K=0, the high Q output of the master goes to the J input of the slave and the Negative transition of the clock sets the slave, copying the master.
5. If J=1 and K=1, it toggles on the positive transition of the clock and thus the slave toggles on the negative transition of the clock.
6. If J=0 and K=0, the flip flop is disabled and Q remains unchanged.

The Master-Slave JK Flip-Flop is a memory element widely used in digital systems. If you want to dive deeper into digital logic and master the flip-flop mechanisms, the GATE CS Self-Paced Course offers detailed explanations and examples to help you understand this important concept.

Timing Diagram of a Master Slave flip flop -



1. When the Clock pulse is high the output of master is high and remains high till the clock is low because the state is stored.
2. Now the output of master becomes low when the clock pulse becomes high again and remains low until the clock becomes high again.
3. Thus toggling takes place for a clock cycle.

4. When the clock pulse is high, the master is operational but not the slave thus the output of the slave remains low till the clock remains high.
5. When the clock is low, the slave becomes operational and remains high until the clock again becomes low.
6. Toggling takes place during the whole process since the output is changing once in a cycle.

This makes the Master-Slave J-K flip flop a Synchronous device as it only passes data with the timing of the clock signal.

Q.16] Explain different memory Mapping Techniques. **5**

Ans:-

Memory Mapping Techniques (Simple & Long Answer)

Memory mapping means **how the CPU accesses memory locations** and how different parts of memory (RAM, ROM, I/O devices) are arranged in the address space.
It tells *where* each device is located in memory and *how* the processor communicates with them.

There are mainly **three important memory mapping techniques**:

---

1. Direct Mapping (Simplest Technique)

Direct mapping is the **easiest and most commonly used method**, especially in caches.

How it works

- Each **block of main memory** is mapped to **only one fixed location in the cache**.
- The memory block number is used to decide where it goes.
- If two memory blocks try to use the same location, one will replace the other.

Features

- Very **simple** hardware
- **Fast** because mapping is direct
- But **conflicts** occur easily

Example

If cache has 8 lines:
Memory block 10 → Line (10 mod 8 = 2)

---

2. Associative Mapping

Associative mapping gives **complete flexibility**.

How it works

- A memory block can be placed in **any line of the cache**.
- There is **no fixed position**.
- The cache checks all lines to find the required block (using tag comparison).

Features

- **No conflict problems**
- Very flexible
- But more **expensive and slower** because all tags must be checked

Example

Block 10 can go to **any line** in the cache.

---

3. Set-Associative Mapping

Set-associative mapping is a **combination of direct and associative mapping**.

How it works

- Cache is divided into **sets**, each set having **2 or more lines**.
- A memory block can be placed **in any line of a specific set**.
- Mapping is fixed to the set but flexible inside the set.

Features

- Much fewer conflicts
- Faster than associative
- More flexible than direct
- Most modern CPUs use this method

Example

For a 2-way set associative cache:
Block 10 → Set (10 mod number_of_sets) and can choose from 2 lines inside that set.

---

4. Memory-Mapped I/O (Important for exams)

In this technique, **I/O devices are treated like memory locations**.

How it works

- CPU uses normal memory instructions (LOAD/STORE) to communicate with devices.
- Each device has a unique address in the memory space.

Features

- Simple and uniform
- No special I/O instructions
- But I/O shares address space with memory

---

5. I/O-Mapped I/O (Isolated I/O)

Here, I/O devices have **a separate address space**, different from memory.

How it works

- Special instructions like **IN** and **OUT** are used to access devices
- Memory and I/O have separate address ranges

Features

- Memory space is not reduced
- Hardware design may be more complex
- Used in microprocessors like 8085

---

6. Paging Memory Mapping (For Virtual Memory)

Paging is used to map **virtual memory to physical memory**.

How it works

- Virtual memory is divided into **pages**
- Physical memory is divided into **frames**
- The **page table** maps every virtual page to a physical frame

Features

- Eliminates fragmentation
- Allows programs larger than RAM
- Used in all modern OS

---

7. Segmentation Mapping

Segmentation divides memory into **logical segments** such as code, data, stack.

How it works

- Each segment has a **base address** and a **limit**

- CPU uses a **segment table** to find the physical address

Features

- Good for user-level program organization
- Supports varying segment sizes

Summary (Easy to Revise)

1. **Direct Mapping** → One block to one fixed cache line
2. **Associative Mapping** → Block can go anywhere
3. **Set-Associative Mapping** → Block goes to one set, but any line inside set
4. **Memory-Mapped I/O** → I/O devices treated as memory
5. **Isolated I/O** → Separate I/O space
6. **Paging** → Virtual pages mapped to physical frames
7. **Segmentation** → Memory is divided into logical segments

Q.17] List and Explain Characteristics Of Memory. **5**

Ans:-

Characteristics of Memory (Simple Explanation)

Memory characteristics describe how a memory device performs and how suitable it is for a computer system. The main characteristics are:

---

1. Storage Capacity

- It tells **how much data** the memory can store.
- Measured in **bits**, **bytes**, **KB**, **MB**, **GB**, etc.
- Higher capacity means more information can be stored.

---

2. Access Time

- It is the **time taken** by the memory to read or write data.
- Shorter access time = faster memory.
- Cache has very low access time, while hard disks have high access time.

---

3. Memory Cycle Time

- Time required to **complete one full read/write cycle** and become ready for the next operation.
- Cycle time ≥ Access time.
- Lower cycle time means better performance.

---

4. Cost per Bit

- Shows how **expensive** the memory is for each stored bit.
- Fast memories (like cache and registers) have **high cost per bit**.
- Slow memories (like HDD) have **low cost per bit**.

---

5. Volatility

- Tells whether the memory keeps data **when power is turned off**.
- **Volatile**: Loses data (RAM).
- **Non-volatile**: Keeps data (ROM, Flash, HDD).

---

6. Physical Size

- Refers to the **space** taken by the memory chip or device.
- Smaller, compact memory is preferred in modern systems.

---

7. Power Consumption

- The amount of **electric power** needed for operation.
- Low power memory is preferred in mobile/portable devices.

---

8. Reliability

- Ability to store data **accurately for a long time**.
- More reliable memory has fewer errors and longer lifetime.

---

9. Bandwidth

- Amount of data that can be **transferred per second**.
- High bandwidth means faster data movement between CPU and memory.

---

10. Organization

- Refers to how memory is **arranged internally**
  (number of lines, rows, columns, banks, word size, etc.)
- Good organization improves performance.

Summary for Revision

- **Capacity** – How much data
- **Access Time** – How fast you can read/write
- **Cycle Time** – Time between two operations
- **Cost per Bit** – Expensive or cheap
- **Volatility** – Data retained or lost
- **Physical Size** – Space taken
- **Power Consumption** – Energy used
- **Reliability** – Error-free operation
- **Bandwidth** – Data transfer rate
- **Organization** – Internal structure

Q.18] What do you mean by cache coherence. **5**

Ans:-

Cache Coherence:-

Cache coherence refers to the **consistency of data stored in multiple caches** in a multiprocessor or multicore system.
In modern computers, each processor or core has its **own private cache** to improve speed.
However, different caches may store **copies of the same memory location**.
When one processor modifies its copy, the other caches may still have the **old value**.
This creates **inconsistent data**, which leads to wrong calculations or incorrect program execution.
The method of keeping all caches **updated, consistent, and synchronized** is called **cache coherence**.

Need for Cache Coherence

Cache coherence is required because:

1. **Multiple caches store the same data** from main memory.
2. **One processor may update the data**, while others continue to use old data.

3. This causes **data conflicts** and errors.
4. Programs that share data between processors must always see the **correct and latest value**. Therefore, cache coherence ensures system **correctness**, **reliability**, and **smooth parallel processing**.

Types of Cache Coherence Problems

1. **Write–Write Conflict**
   Two processors write to the same memory location at different times, causing mismatch.
2. **Read–Write Conflict**
   One processor updates data, while another reads the old value.
3. **Data Replication Problem**
   Same data copied in multiple caches becomes different after updates.

Cache Coherence Mechanisms

To maintain coherence, special techniques are used:

1. Snooping Protocols

- All caches **monitor (snoop)** the shared bus.
- When one cache updates data, others detect it.
- They update or invalidate their copies.
- Simple and used in shared-bus systems.

2. Directory-Based Protocols

- A **directory** keeps track of which cache has which data.
- When a processor updates data, the directory informs all others.
- Used in large multiprocessor systems.

Common Coherence Protocols

MESI Protocol (Most Important)

Each cache block exists in one of four states:

- **M – Modified:** Cache has updated data not in memory
- **E – Exclusive:** Cache has data and no other cache has it
- **S – Shared:** Data is in multiple caches and is valid

- **I – Invalid:** Data is outdated and cannot be used

This prevents stale or conflicting data across caches.

Goals of Cache Coherence

1. **Read Coherence:** Every read gives the most recent value
2. **Write Coherence:** All writes are properly seen by other caches
3. **Consistency:** System behaves the same as if only one cache existed
4. **Correctness:** Parallel programs run accurately

Conclusion

Cache coherence ensures that **all processors work with the same, correct data** in a multiprocessor system.
Without coherence, data becomes inconsistent, leading to **errors and unpredictable behavior**.
With techniques like **snooping, directory protocols, and MESI**, modern systems maintain reliable and synchronized cache data.

Q.19] Explain Concept of Locality of Reference. 05

Ans:-

Locality of reference is an important concept in computer memory systems. It explains how programs actually use memory while they run. When a program is executed, it does not access all memory locations randomly. Instead, it repeatedly uses a small set of instructions and data for some time. Because of this repeated and nearby access, memory systems like cache become very efficient.

Locality of reference mainly means that if a particular memory location is accessed now, then the same location or the nearby locations are likely to be accessed again soon. This behaviour is natural because programs often execute instructions in sequence, and they often work on the same variables or data structures many times. Due to this predictable pattern, faster memory like cache can store recently used data and supply it quickly whenever needed.

There are two main types of locality and one additional related type:

1. Temporal Locality
Temporal locality means re-using the same data or instruction many times within a short period. If a memory location is accessed once, it is likely to be accessed again soon. For example, loop counters, frequently used variables, and repeated instructions inside loops show temporal locality. Cache memory stores these recently used values so that the CPU can access them faster the next time.

2. Spatial Locality
Spatial locality means accessing nearby memory locations. If a program accesses one memory address, it often accesses the next memory address soon. This happens because instructions in a program are stored sequentially, and arrays or data structures are stored in adjacent memory. Cache

uses this property by bringing an entire block of nearby addresses at once so the CPU gets faster access to future data.

3. Sequential Locality
Sequential locality is a special case of spatial locality where instructions or data are accessed in a strict sequential order. Many programs execute statements one after another unless there is a branch or jump. Thus, reading the next instruction is predictable. This makes instruction prefetching possible.

Locality of reference is extremely important for the design of memory hierarchy. It allows systems to use small but fast memory (like cache) effectively. By keeping recently used and nearby data in the cache, the CPU saves time and reduces the number of expensive main memory accesses. Locality is also used in virtual memory, paging, TLB design, replacement algorithms, and overall system performance improvements.

In conclusion, locality of reference explains how programs naturally access memory in repeated and nearby patterns. It includes temporal locality, spatial locality, and sequential locality. This concept helps memory systems work faster by predicting what data the program will use next. It is one of the core principles behind caching and efficient memory hierarchy in modern computers.

Difference/Compare

Q.1] Explain the difference between a Multiplexer and Demultiplexer with suitable parameters.10

Ans:-

Multiplexer vs Demultiplexer – 4 Column Table (15 Points)

Sr. No.	**Parameter**	**Multiplexer (MUX)**	**Demultiplexer (DEMUX)**
1	Basic Function	Many inputs → one output	One input → many outputs
2	Data Flow Direction	Combines signals	Distributes signals
3	Speed	Faster (just selects one input)	Slightly slower (routes to many outputs)
4	Efficiency	Saves bandwidth by merging	Efficient in parallel distribution
5	Select Lines	Used to choose input	Used to choose output
6	Number of Lines	Many inputs, single output	Single input, many outputs
7	Circuit Complexity	More input gating	More output gating
8	Cost	Slightly higher	Slightly lower
9	Hardware Used	AND/OR gates	AND/NOT gates
10	Data Transmission	Many → single channel	Single → multiple channels
11	Application Area	Communication systems	Decoder, routers

Sr. No.	Parameter	Multiplexer (MUX)	Demultiplexer (DEMUX)
12	Control Signals	Only select lines	Select + enable
13	Operation Nature	Works as data selector	Works as data distributor
14	Reverse Operation	Reverse of DEMUX	Reverse of MUX
15	Example Use	Telephone network combining	Broadcasting to multiple devices

Q.2] Explain microprogrammed control unit and hardwired control units,

Compare microprogrammed and hardwired control units,

List advantages/disadvantages. 10

Ans:- Part A:-

Long Note: Microprogrammed Control Unit and Hardwired Control Unit

A control unit is an essential part of the CPU responsible for generating all the control signals required to execute an instruction. It directs the flow of data between the CPU, memory, and input/output devices by coordinating operations like fetching, decoding, and executing instructions. There are two major approaches used to design a control unit: **Microprogrammed Control Unit** and **Hardwired Control Unit**. Both methods aim to generate control signals, but they differ greatly in how they are designed, how they operate, and how easily they can be modified.

Microprogrammed Control Unit

A **microprogrammed control unit** uses a set of microinstructions stored in a special memory called the **Control Store**. Instead of generating control signals directly using hardware, it reads these microinstructions like a small program. Each microinstruction contains the necessary control signals for performing one or more micro-operations. The entire collection of microinstructions for all the CPU instructions is called a **microprogram**.

This approach is very **flexible** because modifying or adding instructions only requires updating the microprogram, not the hardware. It is commonly used in **Complex Instruction Set Computers (CISC)**, where instructions have many steps and require detailed control. Although this method is easier to implement and change, it is **slower** compared to hardwired control because fetching microinstructions from memory takes time. Still, it is very useful for systems that need to support many complex instructions, emulation, or backward compatibility.

Hardwired Control Unit

A **hardwired control unit** generates control signals using fast and dedicated hardware components such as combinational logic circuits, decoders, counters, and flip-flops. Here, the sequence of operations needed to execute an instruction is built directly into the hardware's wiring and logic design. Because the signals are produced immediately by circuits, this type of control unit is **very fast** and preferred in **Reduced Instruction Set Computers (RISC)**, where speed is essential.

Although hardwired control units offer high performance, they are **difficult to modify**. Any change in instruction format or timing requires redesigning parts of the hardware, which is costly and time-consuming. They are best suited for simple instruction sets where speed and efficiency are more important than flexibility.

Part B&C:-

Sr. No.	**Parameter**	**Hardwired Control Unit**	**Microprogrammed Control Unit**
1	Basic Concept	Control signals generated using **combinational logic**	Control signals generated using **microinstructions stored in memory**
2	Speed	**Very fast**	**Slower** compared to hardwired
3	Flexibility	Less flexible	Highly flexible
4	Modification	Difficult to modify	Easy to modify by changing microcode
5	Design Complexity	Complex logic design	Simpler design
6	Cost	More expensive due to hardware	Cheaper because uses memory
7	Implementation	Uses **decoders, gates, flip-flops**	Uses **control memory + microprograms**
8	Reliability	High reliability	Slightly less due to memory access
9	Instruction Set	Suitable for **RISC** processors	Commonly used in **CISC** processors
10	Control Signal Generation	Generated directly by hardware	Generated sequentially from micro-ops
11	Execution Time	Constant and very quick	Depends on microinstruction cycle
12	Error Fixing	Hard to correct errors	Easy to update microcode
13	Speed Dependency	Depends on logic propagation delays	Depends on microinstruction memory speed
14	Use Case	Fast processors requiring high performance	Processors needing complex instructions

Sr. No.	**Parameter**	**Hardwired Control Unit**	**Microprogrammed Control Unit**
15	Example	Old RISC CPUs	Intel 8086, 8088 (CISC)

Q.3] Differentiate computer organization vs. computer architecture. 10

Ans:-

Computer Organization vs Computer Architecture (15 Points – Table)

Sr. No.	**Parameter**	**Computer Organization**	**Computer Architecture**
1	Basic Meaning	How the system **works internally**	How the system is **designed logically**
2	Focus	Implementation of hardware	Function, structure, and behavior
3	Deals With	Physical components	Programming and design concepts
4	Viewpoint	“How to do it”	“What to do”
5	Example Elements	ALU, registers, control signals	Instruction set, addressing modes
6	User Visibility	Not visible to programmer	Visible to programmer
7	Instruction Set	Does **not** define ISA	Defines the Instruction Set Architecture (ISA)
8	Performance Impact	Affects execution speed directly	Affects system capabilities
9	Implementation	Actual hardware implementation	High-level design of hardware
10	Flexibility	Less flexible—hardware dependent	More flexible—logical design based
11	Modification	Hard to modify (hardware changes)	Easier to modify (ISA changes)
12	Speed Dependency	Depends on hardware components	Depends on instruction design
13	Seen In	Microarchitecture level	System architecture level

Sr. No.	Parameter	Computer Organization	Computer Architecture
14	Goal	Efficient hardware operation	Efficient program execution
15	Example	Size of RAM, bus structure	RISC, CISC design principles

Q.4] Differentiate Interleaved and Associative Memory. **10**

Ans:-

Interleaved Memory vs Associative Memory (15 Points – Table)

Sr. No.	**Parameter**	**Interleaved Memory**	**Associative Memory (Content Addressable Memory)**
1	Basic Meaning	Memory divided into **multiple modules**	Memory where data is accessed by **content**, not address
2	Access Method	Access by **address**	Access by **data/content**
3	Working	Parallel access from different memory banks	Searches all locations simultaneously
4	Speed	Increases speed by overlapping accesses	Very fast search operations
5	Purpose	Improve **throughput**	Improve **search efficiency**
6	Structure	Sequential memory units	Fully parallel comparison circuits
7	Access Time	Reduced due to parallelism	Constant time because all entries checked at once
8	Use Case	High-speed processors, pipelining	Cache tags, TLBs, lookup tables
9	Flexibility	Used only for normal address access	Used for matching patterns/data
10	Cost	Cheaper, simpler design	Expensive due to parallel hardware
11	Hardware Complexity	Low	High (parallel comparators)
12	Example	4-way or 8-way interleaving in RAM	CAM (Content Addressable Memory)

Sr. No.	Parameter	Interleaved Memory	Associative Memory (Content Addressable Memory)
13	Access Type	Address-based read/write	Content-based read
14	Suitable For	Increasing memory bandwidth	Fast searching and mapping
15	Memory Organization	Multi-bank memory system	Associative array memory system

Q.5] Compare SRAM vs DRAM with parameters. **10**

Ans:-

SRAM vs DRAM (15 Points – Table)

Sr. No.	Parameter	SRAM (Static RAM)	DRAM (Dynamic RAM)
1	Full Form	Static Random Access Memory	Dynamic Random Access Memory
2	Storage Method	Uses **flip-flops**	Uses **capacitors**
3	Refreshing	**No refresh required**	**Requires periodic refresh**
4	Speed	Very **fast**	**Slower** than SRAM
5	Cost	**Expensive**	**Cheaper**
6	Density	Low density (less data per chip)	High density (more storage per chip)
7	Power Consumption	Consumes **more power**	Consumes **less power**
8	Bit Cell Size	Larger (6 transistors per cell)	Smaller (1 transistor + capacitor)
9	Complexity	More complex circuit	Simpler circuit
10	Access Time	Low (fast access)	Higher (slower access)
11	Usage	Used in **cache memory**	Used in **main memory (RAM)**
12	Reliability	Very reliable	Less reliable due to charge leak
13	Data Retention	Holds data as long as power is ON	Charge leaks → must be refreshed
14	Manufacturing Cost	High manufacturing cost	Low manufacturing cost
15	Example	L1/L2/L3 CPU cache	Computer system RAM sticks

Q.6] Distributed vs Centralized Bus Arbitration. **10**

Ans:-

Distributed vs Centralized Bus Arbitration (15 Points – Table)

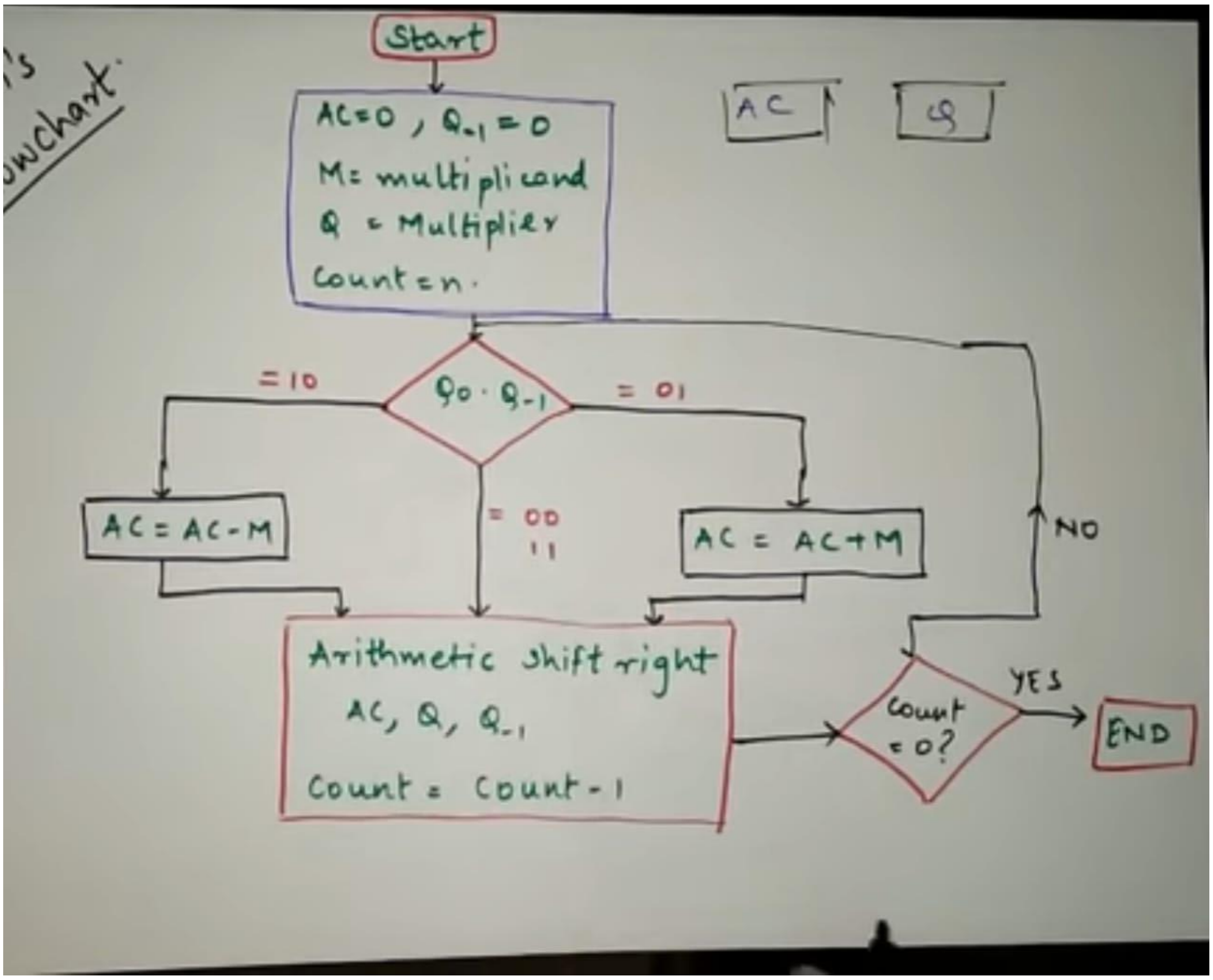
Sr. No.	**Parameter**	**Centralized Bus Arbitration**	**Distributed Bus Arbitration**
1	Basic Idea	One **single master** controls bus access	Control is **shared among all devices**
2	Arbiter Location	**Central arbiter** present	No central arbiter
3	Control Method	Master grants/denies bus request	Devices cooperate & decide among themselves
4	Speed	Faster decision-making	Slightly slower due to coordination
5	Complexity	Simple design	More complex logic
6	Reliability	Failure of master → system stops	No single failure point (more reliable)
7	Cost	Cheaper due to single arbiter	Costly (each device needs arbitration logic)
8	Priority Handling	Priority fixed by central master	Priority decided dynamically by devices
9	Scalability	Less scalable (master becomes bottleneck)	Highly scalable (each device participates)
10	Communication Overhead	Low	Higher due to distributed negotiation
11	Hardware Requirement	Only one arbiter circuit	Arbiter logic in every device
12	Use Cases	Small computers, simple systems	Multiprocessor systems, advanced networks
13	Example	Daisy-chaining, centralized parallel arbiter	IEEE 1394 (FireWire), multi-master buses
14	Conflict Resolution	Arbiter resolves conflicts	Devices communicate to resolve conflicts

Sr. No.	Parameter	Centralized Bus Arbitration	Distributed Bus Arbitration
15	Performance	Good for small systems	Best for large, multi-processor systems

Numericals & Diagrams

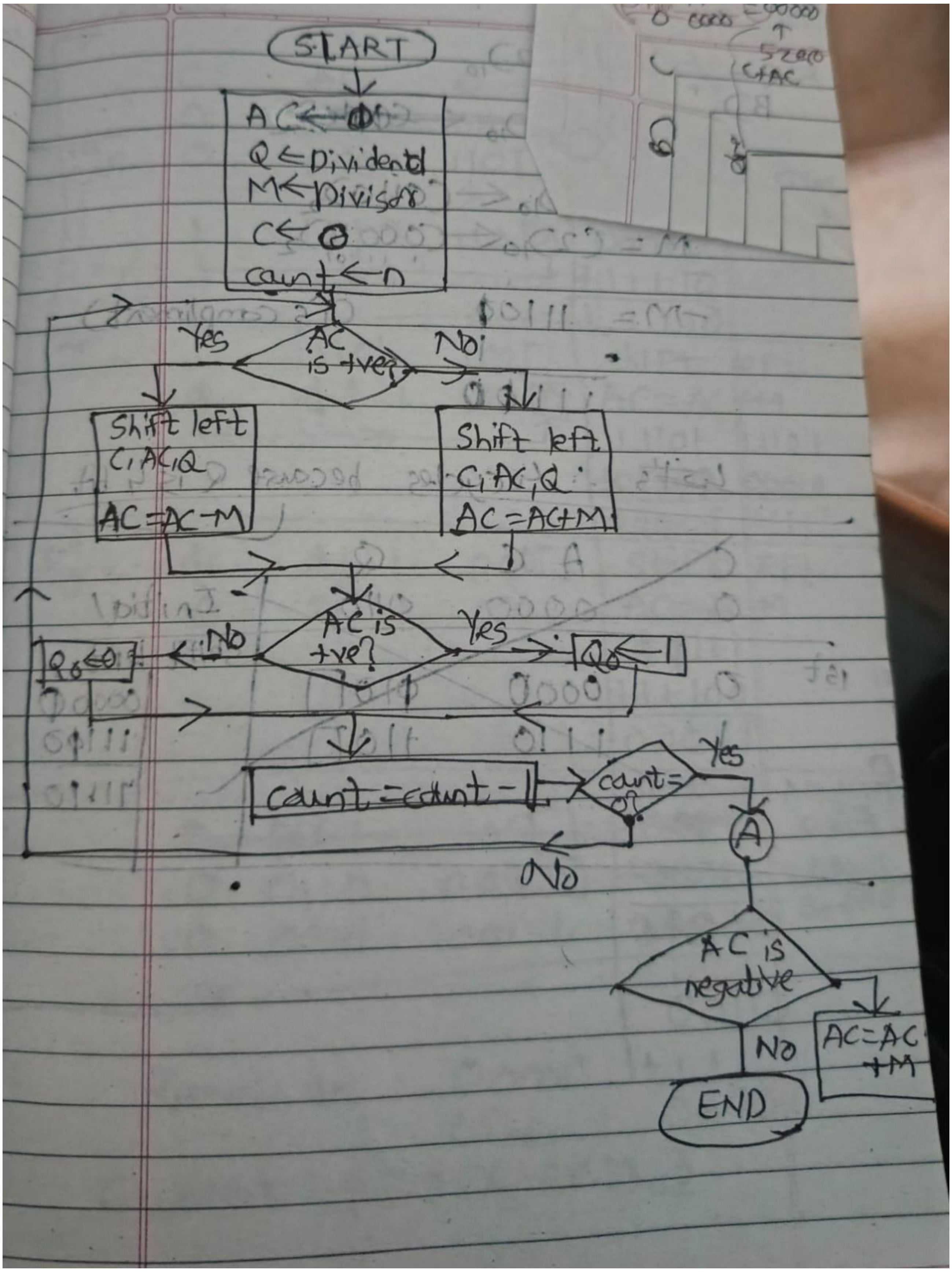
Q1] Booth's Algorithm. **10**

Ans:-



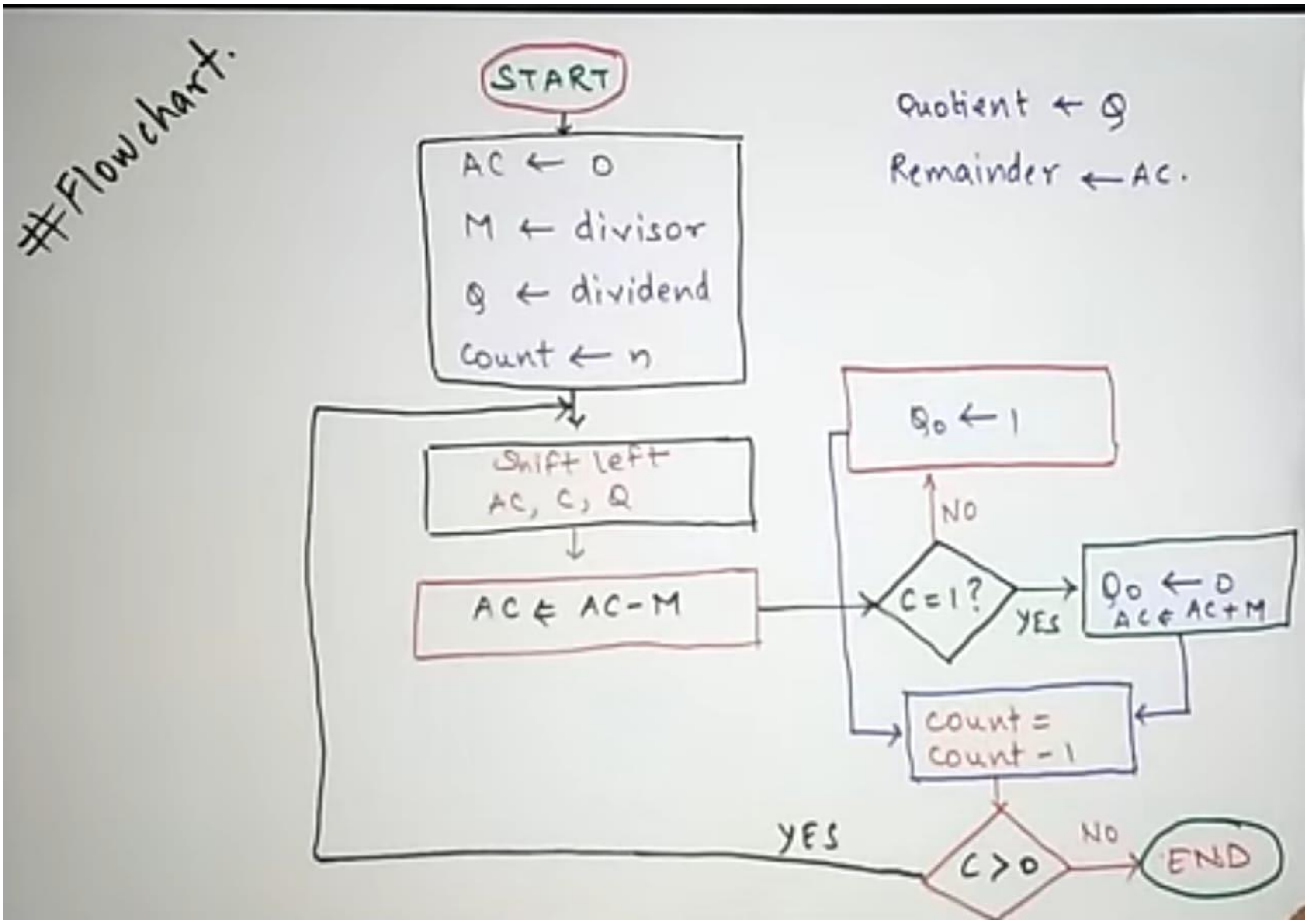
Q2] Non-Restoring Algorithm. **10**

Ans:-



Q3] Restoring Algorithm. **10**

Ans:-


#Flowchart.
START
AC ← 0
M ← divisor
Q ← dividend
Count ← n
Shift left AC, C, Q
AC ← AC−M
C=1?
NO
YES
Q0 ← 1
Q0 ← 0
AC ← AC+M
count = count − 1
C>0
YES
NO
END
Quotient ← Q
Remainder ← AC.