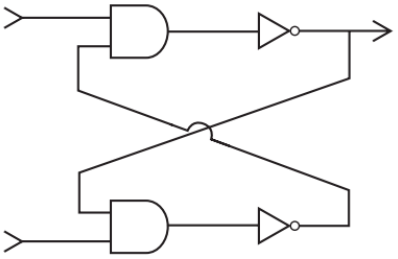
Scheme of Evaluation

Internal Assessment Test 1 – November 2025

Sub:	Essentials of IT					**Sub Code:**	1BESC104E	**Branch:**	Basic Science
Date:	07/11/2025	**Duration:**	90 mins	**Max Marks:**	50	**Sem**	I		

Question #	**Description**	**Marks Distribution**	**Max Marks**
1a	**List the three main components of the CPU and explain the von Neumann architecture.**		5
	CPU Components and description Explanation of Von Neumann Architecture	3M 2M	
1b	**Define the fundamental role of the Arithmetic/Logic Unit (ALU) in a computer.**		5
	Definition of ALU Explanation of arithmetic and logical operations Role in data handling and interaction with registers	1.5M 1.5M 2M	
2	**Explain what happens in the given circuit when the upper input changes from 1 to 0 and when the lower input changes from 1 to 0. Redraw the circuit using NAND gates.** 		10
3	**Summarize the steps involved in the booting procedure of a computer system.**		10
	What is booting Explanation of ROM and bootloader Explanation of OS Transfer Explanation of Transfer of Control Role of BIOS/EFI	2M 2M 2M 2M 2M	
4	**Solve the following problems by converting the given decimal values into 5-bit two's complement form and performing the operations:** **a) 5 + 1** **b) 5 − 1**		10
	Conversion of decimal to binary Binary addition Final result	2M * 2 2M * 2 1M * 2	
5	**Develop a pseudocode algorithm for giving directions in a subject you are familiar with. Describe the primitives you would use and specify the syntax rules you would define.**		10

	Concept of algorithm representation Definition of primitives Examples of primitives Syntax rules Pseudocode	1M 2M 2M 2M 3M	
6a	**Explain the differences between geometric and bitmap image representation methods used in computer graphics.**		5
	Definition of geometric representation Definition of bitmap representation Differences	1M 1M 3M	
6b	**Convert the following decimal fractions into binary notation:** **a) 5¾ b) 5⅜**		5
	Binary conversion of integer part Binary conversion of fractional part Final answer	1M * 2 1M * 2 0.5M * 2	
7	**Define process priority and state how it affects process execution speed in a multiprogramming system.**		10
	Definition of process priority Role of scheduler and dispatcher Effect on CPU allocation Effect on waiting time Diagram/Example	2M 2M 2M 2M 2M	

Solutions

1a. List the three main components of the CPU and explain the von Neumann architecture.

The Central Processing Unit (CPU) is the main component responsible for performing all calculations and decision-making in a computer system. It is often referred to as the brain of the computer because it manages and coordinates all operations. The CPU in modern computers is usually a microprocessor, which is a single chip containing millions of transistors.

The CPU is made up of three main components:

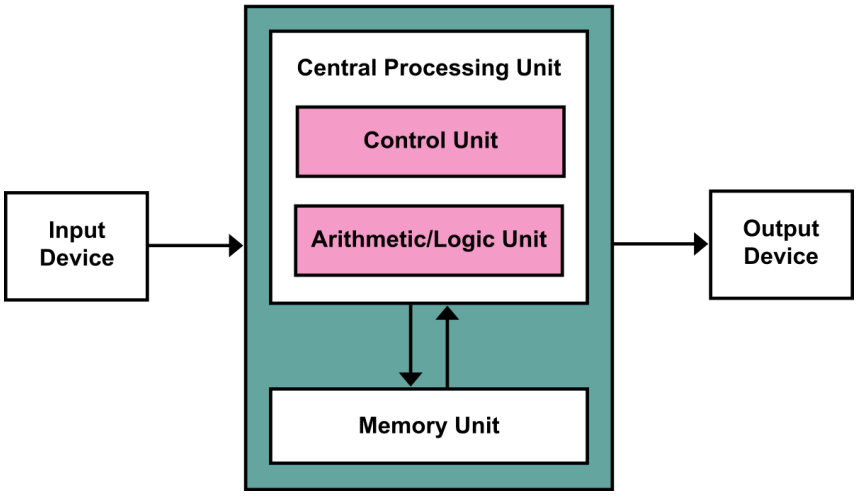
- Arithmetic/Logic Unit (ALU)
- Control Unit (CU)
- Register Unit

The Arithmetic/Logic Unit (ALU) carries out all arithmetic and logical operations. Arithmetic operations include addition, subtraction, multiplication, and division, while logical operations involve comparisons and conditions such as AND, OR, NOT, and greater than or equal to checks. The ALU can be thought of as the calculator within the CPU.

The Control Unit (CU) directs and manages the operations of the CPU. It controls the flow of data between the CPU, memory, and input/output devices. The CU sends control signals to the ALU and registers, instructing them on when and how to carry out operations.

The Register Unit contains small, high-speed memory locations inside the CPU. Registers temporarily store data, instructions, and intermediate results that the CPU is actively working with. Important registers include the Instruction Register (IR), which holds the current instruction being executed, and the Program Counter (PC), which keeps track of the address of the next instruction.

The von Neumann architecture was proposed by John von Neumann in 1945. In this architecture, both data and program instructions are stored together in the same main memory. The CPU retrieves instructions and data from memory through a shared communication channel called the system bus. Execution follows a defined sequence known as the fetch-decode-execute cycle, where the CPU fetches an instruction from memory, decodes it to understand the required operation, executes it, and stores the result. This design allows the CPU to process instructions sequentially and efficiently, forming the basic operational model of modern computers.



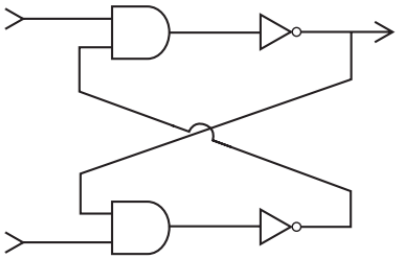
1b. Define the fundamental role of the Arithmetic/Logic Unit (ALU) in a computer.

The Arithmetic/Logic Unit (ALU) is a core component of the Central Processing Unit responsible for performing all arithmetic and logical operations within a computer system. It acts as the part of the CPU that actually carries out the mathematical and decision-making tasks required during instruction execution.

The ALU performs two main categories of operations: arithmetic and logical. Arithmetic operations include basic mathematical calculations such as addition, subtraction, multiplication, and division. Logical operations involve decision-based comparisons such as AND, OR, NOT, and equality or inequality checks. These logical functions allow the computer to evaluate conditions and make decisions during program execution.

In addition to performing computations, the ALU plays an essential role in handling data and interacting with the CPU's registers. The registers temporarily store input data, intermediate results, and final outputs of ALU operations. The Control Unit sends signals to the ALU, specifying which operation to perform and which registers to use. After completing an operation, the ALU stores the result back into a register or sends it to main memory for further use. Through this continuous interaction with the Control Unit and registers, the ALU enables the CPU to execute instructions efficiently and maintain smooth data processing within the system.

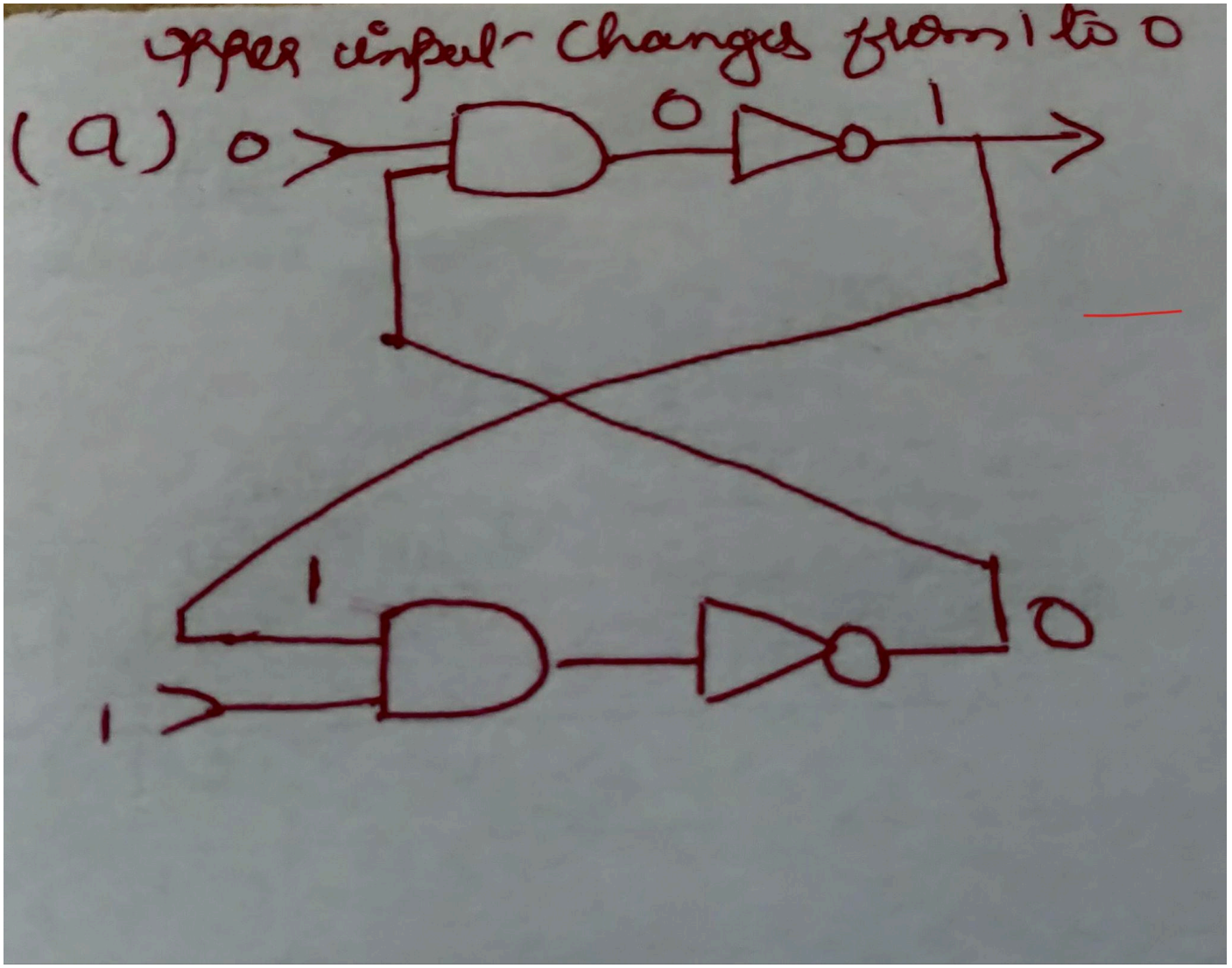
2. Explain what happens in the given circuit when the upper input changes from 1 to 0 and when the lower input changes from 1 to 0. Redraw the circuit using NAND gates.



1. If the upper input is temporarily changed to 0:

- The upper AND gate receives inputs 0 and the feedback from the lower side.
- Since one input is 0, the output of the upper AND gate becomes 0.
- The inverter after the upper AND gate outputs 1.
- This 1 is fed back to the lower AND gate.
- The lower AND gate now has inputs 1 (lower input) and 1 (feedback), so its output is 1.
- The inverter after the lower AND gate outputs 0.
- The output of the lower side becomes 0.

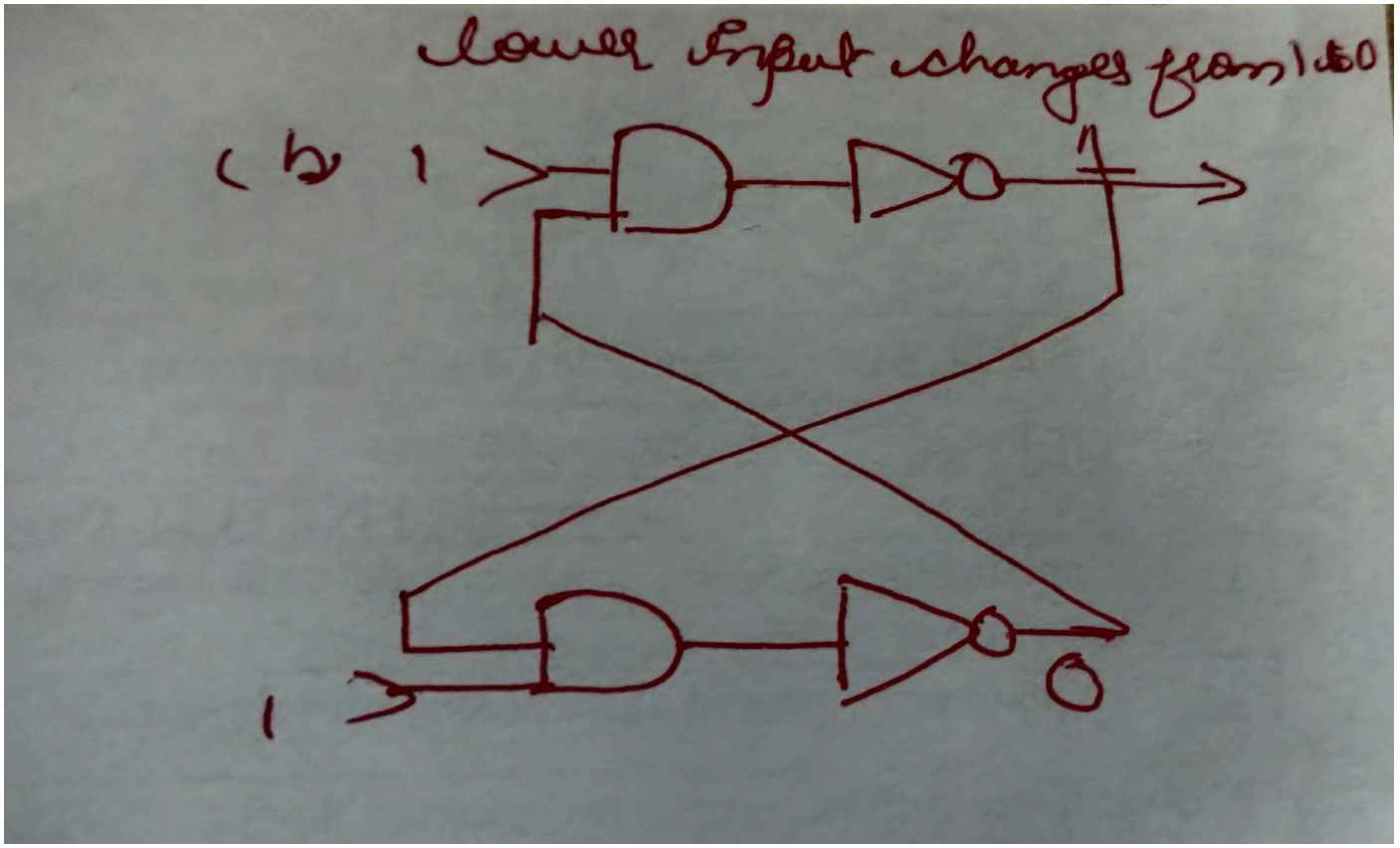
Result: The latch sets the lower output to 0 and the upper output to 1, storing a bit accordingly.



2. If the lower input is temporarily changed to 0:

- The lower AND gate receives inputs 0 and the feedback from the upper side.
- Since one input is 0, the output of the lower AND gate becomes 0.
- The inverter after the lower AND gate outputs 1.
- This 1 is fed back to the upper AND gate.
- The upper AND gate now has inputs 1 (upper input) and 1 (feedback), so its output is 1.
- The inverter after the upper AND gate outputs 0.
- The output of the upper side becomes 0.

Result: The latch sets the upper output to 0 and the lower output to 1, storing a bit accordingly.



3. Summarize the steps involved in the booting procedure of a computer system.

The process of starting a computer and loading the operating system into main memory is known as booting or bootstrapping. It begins when the system is powered on and continues until the operating system takes control of the computer's operations. Booting ensures that all the essential software components are properly initialized and ready for use.

When a computer is switched on, the contents of its main memory are empty because it is a volatile type of storage that loses all data when power is off. To solve this, a small section of memory is made from Read-Only Memory (ROM), which is non-volatile and retains its contents permanently. Stored within ROM is a small program called the boot loader, which is the very first code executed by the CPU after power-up. The boot loader's purpose is to start the process of bringing the operating system into memory.

The next step involves transferring the operating system from a permanent storage device into main memory. The boot loader contains instructions that guide the CPU to locate the operating system stored on a hard drive, solid-state drive, or sometimes over a network. Once located, the boot loader loads the necessary operating system files into main memory so that the CPU can execute them directly. This process prepares the system to become operational.

After the operating system has been successfully loaded into memory, the boot loader performs a transfer of control. This is achieved by executing a jump instruction that hands over control from the boot loader to the operating system. From this point onward, the operating system becomes responsible for managing the computer's hardware and software resources, handling user input, and running applications.

Throughout the booting process, the boot loader uses firmware routines such as the BIOS (Basic Input/Output System) or EFI (Extensible Firmware Interface). These routines are stored in non-volatile memory and provide the basic input and output functions required before the operating system becomes active. The BIOS or EFI helps the system check connected devices, initialize hardware components, and establish the basic communication required for loading the operating system. Together, these steps ensure a smooth and reliable transition from powered-off hardware to a fully functional operating system environment.

4. Solve the following problems by converting the given decimal values into 5-bit two's complement form and performing the operations:

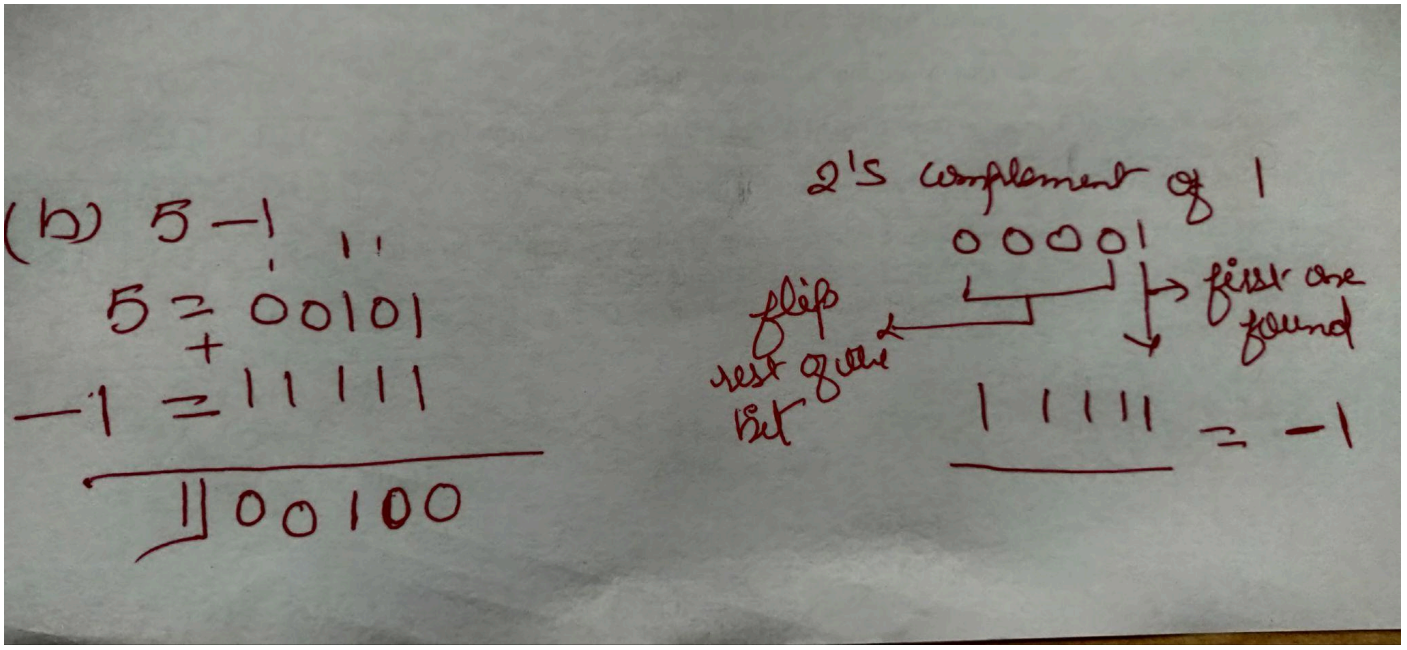
a)5 + 1

5 in decimal = 00101
1 in decimal = 00001

```
   00101
+  00001
-------
   00110
```

Sign bit = 0 = Positive 0110 in decimal = 6
Final answer = +6

b) 5 − 1



5. Develop a pseudocode algorithm for giving directions in a subject you are familiar with. Describe the primitives you would use and specify the syntax rules you would define.

START
  // Input: starting point (Your House), destination (Bus Stop)

```
    READ StartingPoint
    READ Destination

    OUTPUT "You are at " + StartingPoint
    OUTPUT "We will guide you to " + Destination

    // Step 1: Exit the house
    OUTPUT "Exit your house and walk to the front gate."

    // Step 2: Walk to the main road
    OUTPUT "Walk straight ahead for about 50 metres until you reach the main road."

    // Step 3: At main road, decide direction
    IF MainRoadDirection = "left" THEN
        OUTPUT "Turn left onto the main road."
    ELSE
        OUTPUT "Turn right onto the main road."
    ENDIF

    // Step 4: Continue until landmark
    OUTPUT "Continue walking straight for 200 metres until you see the red building on your right."

    // Step 5: Landmark reached — bus stop ahead
    OUTPUT "At the red building, turn right and walk 20 metres."
    OUTPUT "The bus stop is on your left."

    // Step 6: Arrival
    OUTPUT "You have arrived at " + Destination

END
```

Explanation:
The algorithm begins by reading the starting point and destination (though in this simple case they're known).

It then outputs a sequence of directions: exit the house, walk straight, turn onto a road, continue until a landmark, then final turn and arrival.

There is a conditional (IF) to decide which way to turn onto the main road (depending on which side the main road is).

Each step is one statement per line, using OUTPUT to display instructions.

The syntax rules above ensure clarity and consistency.

4. If I wanted to make it more general (say: giving directions between any two points on a map), I could expand it with loops and sub‑routines. For example:

```
PROCEDURE GiveDirections(StartPoint, EndPoint)
    OUTPUT "You are at " + StartPoint
    OUTPUT "Destination: " + EndPoint
```

```
  SET CurrentPoint = StartPoint

  WHILE CurrentPoint ≠ EndPoint DO
    // Determine next segment
    SET NextSegment = DetermineNextSegment(CurrentPoint, EndPoint)
    OUTPUT "Proceed to " + NextSegment.Landmark
    IF NextSegment.Turn = "left" THEN
      OUTPUT "Turn LEFT at " + NextSegment.Landmark
    ELSE
      OUTPUT "Turn RIGHT at " + NextSegment.Landmark
    ENDIF
    SET CurrentPoint = NextSegment.NextPoint
  ENDWHILE

  OUTPUT "You have arrived at " + EndPoint
END PROCEDURE
```

6a. Explain the differences between geometric and bitmap image representation methods used in computer graphics.

Attribute	Bitmap / Raster	Geometric / Vector
Representation	Grid of pixels; each pixel stores colour data.	Mathematical primitives (points, lines, curves).
Scaling / Resolution Incident	Fixed resolution; zooming in can cause pixelation (blocks visible).	Resolution independent (in many cases); shapes redraw smoothly at any size.
File size behaviour	File size tends to increase with resolution & pixel count.	File size often lower for comparable simple images, since you store formulas not individual pixels.
Suitability for detail	Very good for natural‑photographic images with complex colour gradations, textures, subtle shading.	Very good for logos, icons, illustrations, diagrams—i.e., shapes, curves, where crisp edges matter.
Editing/Manipulation	Editing is often pixel by pixel (colour changes, retouching).	Editing is manipulating shapes/objects (move a curve, change fill/stroke).

Memory/Computational Cost	Potentially large memory if high resolution, many colours.	Less memory for simple graphics; may require more computation when rendering complex vectors at high detail.
Rasterization / Final Display	Naturally maps to pixel display devices.	Often must be "rasterized" (converted into pixels) for display on screen or print.
Loss & compression	Can use lossy or lossless compression; scaling up is problematic.	Generally lossless in scaling; but converting to raster or exporting may introduce loss.

6b. Convert the following decimal fractions into binary notation: a) 5¾ b) 5⅜

a. Convert 5 to binary by repeatedly dividing by 2 = 101
   Convert 0.75 to binary = 11

   Answer = 101.11

b. Convert 5 to binary = 101
   Convert ⅜ to binary = 011

   Answer = 101.011

7. Define process priority and state how it affects process execution speed in a multiprogramming system.

Process priority is a value assigned by the operating system to determine the order in which processes are executed by the CPU. It represents the relative importance of each process, allowing the system to decide which task should receive more immediate attention. A process with higher priority is executed before those with lower priority, ensuring that critical or time-sensitive tasks are completed faster.

In a multiprogramming system, the scheduler and dispatcher are key components that use process priority to manage CPU time. The scheduler maintains a list of all processes in a process table, recording each process's state (ready, waiting, or running) along with its priority level. Based on this information, the scheduler selects the highest-priority process that is ready to run. The dispatcher then handles the actual transfer of control, loading the selected process into the CPU for execution and managing context switches between processes.

CPU allocation is directly influenced by priority levels. The dispatcher always assigns the CPU to the process with the highest priority in the ready queue. If a new process with a higher priority becomes ready while another process is running, the system may preempt the current process and allocate the CPU to the higher-priority one. As a result, high-priority processes gain faster access to CPU time and execute more quickly, while low-priority processes may experience delays.

Process priority also affects the waiting time of processes. High-priority tasks generally have shorter waiting times because they are executed sooner, whereas low-priority tasks may have to wait longer in the queue. In extreme cases, low-priority processes might suffer from starvation if higher-priority processes continuously

occupy the CPU. To prevent this, many systems use an aging technique, which gradually increases the priority of processes that have been waiting for a long time, ensuring fairness in CPU access.

An example can help illustrate this concept. Suppose three processes P1, P2, and P3 have priorities 3, 1, and 5 respectively (where a higher number means higher priority). In this case, the CPU will first execute P3, then P1, and finally P2. This ensures that P3, being the most critical, completes its task earliest.