**CS3351-Digital Principles and Computer Organization**

**UNIT 1 COMBINATIONAL LOGIC**

### ****1: Combinational Circuits - Karnaugh Map and Binary Adder/Subtractor****

#### ****Answer:****

**Karnaugh Map (K-map) Analysis and Design Procedure:**

* **K-map** is a graphical method used to simplify Boolean expressions and find minimal sum-of-products (SOP) or product-of-sums (POS) forms.
* **K-map for 2, 3, or 4 variables**: A 2-variable K-map is a 2x2 grid, a 3-variable K-map is a 2x4 grid, and a 4-variable K-map is a 4x4 grid.

##### **K-map Simplification Steps:**

1. Write the Boolean expression or truth table.
2. Draw the K-map grid.
3. Place 1s or 0s based on the truth table.
4. Group adjacent 1s (or 0s) into rectangles (powers of 2).
5. Write the simplified Boolean expression from the groups.

##### **Example:**

For the Boolean expression:  
F(A, B, C, D) = Σ(0, 1, 2, 3, 7, 8, 9, 12)

Simplify using a K-map for four variables. The K-map grid would look like:

| **AB \ CD** | **00** | **01** | **11** | **10** |
| --- | --- | --- | --- | --- |
| **00** | 1 | 1 | 0 | 1 |
| **01** | 1 | 1 | 0 | 1 |
| **11** | 0 | 0 | 0 | 0 |
| **10** | 1 | 1 | 1 | 0 |

Simplify the groups, and you'll obtain the minimal Boolean expression.

### ****4-bit Binary Adder Design:****

* A **4-bit binary adder** is created using **Full Adder** circuits. A full adder adds three bits (A, B, and carry-in) and produces a sum and carry-out.
* **Block Diagram**:
  + Four full adders are connected in series.
  + The carry-out from one full adder becomes the carry-in for the next.

**Truth Table for Full Adder:**

| **A** | **B** | **Cin** | **Sum** | **Cout** |
| --- | --- | --- | --- | --- |
| 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 |

##### **4-bit Adder Circuit**:

* For a 4-bit binary adder, the carry-out of each full adder becomes the carry-in of the next full adder in the sequence.

### ****Binary Subtractor:****

* A **binary subtractor** can be constructed using a 4-bit adder and a **2's complement** operation.
* To subtract two numbers, A and B (A - B), use the identity:  
  A - B = A + (2’s complement of B).

##### **2's complement of B**:

* Invert each bit of B and add 1 to the result.

**Circuit Design**:

1. Use a full adder for the binary addition.
2. XOR each bit of B with the control signal (M). If M = 1, perform the 2’s complement on B.
3. The final result of the adder will be the difference between A and B.

### ****Question 2: Decimal Adder and Magnitude Comparator****

#### ****Question:****

* Explain the design of a **Decimal Adder** (BCD adder) and the design of a **2-bit Magnitude Comparator**. Discuss the logic behind each and draw the relevant truth tables.

#### ****Answer:****

### ****Decimal Adder (BCD Adder)****:

A **BCD Adder** is used for adding two decimal numbers where each decimal digit is represented in binary-coded decimal (BCD) format. Each BCD digit uses 4 bits.

#### ****BCD Addition Process****:

1. **Add the two BCD digits** using a 4-bit binary adder.
2. **Check the sum**: If the sum is greater than 9 (1001 in binary), add 6 (0110 in binary) to correct the result into valid BCD.
3. **Carry Handling**: If there’s a carry from the sum, add it to the next BCD digit.

#### ****BCD Adder Example****:

* Add two BCD numbers: 9 (1001) and 7 (0111).
  + Binary sum = 1001 + 0111 = 0000 1 (carry-out).
  + Since the sum exceeds 9 (1001), add 0110 to the result:
  + Correct sum = 0000 1 + 0110 = 0110 (which is 16 in decimal).

### ****2-bit Magnitude Comparator****:

A **2-bit Magnitude Comparator** compares two 2-bit binary numbers (A1A0 and B1B0) and determines if:

* A > B
* A = B
* A < B

#### ****Truth Table for 2-bit Magnitude Comparator****:

| **A1** | **A0** | **B1** | **B0** | **A>B** | **A=B** | **A<B** |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 | 0 |

The magnitude comparator circuit can be implemented using **logic gates** and **XOR** gates for comparison.

**UNIT II: SYNCHRONOUS SEQUENTIAL LOGIC**

**13-Mark Question 1: Introduction to Sequential Circuits and Flip-Flops**

**Question:**

Explain **sequential circuits** and discuss the operation of **flip-flops** (SR, D, JK, T). Also, provide the **excitation tables** for each type of flip-flop.

**Answer:**

**Introduction to Sequential Circuits:**

* **Sequential Circuits**: Unlike combinational circuits, sequential circuits have memory elements (like flip-flops) that store information, meaning the output depends on the current inputs **and** the stored state (previous history).
  + **Memory**: These circuits can remember past events, which is why they are called "sequential."
  + **Clocking**: In **synchronous sequential circuits**, changes in states are triggered by a clock signal (a pulse). The clock governs when the flip-flops will change state, making it predictable and synchronized.

**Types of Flip-Flops:**

1. **SR (Set-Reset) Flip-Flop**:
   * **Inputs**: S (Set), R (Reset)
   * **Outputs**: Q, Q' (complement of Q)
   * **Operation**:
     + S = 1, R = 0 → Set state: Q = 1.
     + S = 0, R = 1 → Reset state: Q = 0.
     + S = 0, R = 0 → Retain previous state.
     + S = 1, R = 1 → Invalid state (both inputs are 1 simultaneously).

**Excitation Table for SR Flip-Flop**:

| **Current State (Q)** | **Next State (Q')** | **S** | **R** |
| --- | --- | --- | --- |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 |

1. **D (Data) Flip-Flop**:
   * **Inputs**: D (Data), Clock (clk)
   * **Outputs**: Q, Q'
   * **Operation**: Q takes the value of D at the rising/falling edge of the clock.
   * If D = 0, Q will be 0; if D = 1, Q will be 1.

**Excitation Table for D Flip-Flop**:

| **Current State (Q)** | **Next State (Q')** | **D** |
| --- | --- | --- |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

1. **JK Flip-Flop**:
   * **Inputs**: J, K, Clock (clk)
   * **Outputs**: Q, Q'
   * **Operation**:
     + J = 0, K = 0 → No change (hold).
     + J = 0, K = 1 → Reset (Q = 0).
     + J = 1, K = 0 → Set (Q = 1).
     + J = 1, K = 1 → Toggle (Q changes to the opposite state).

**Excitation Table for JK Flip-Flop**:

| **Current State (Q)** | **Next State (Q')** | **J** | **K** |
| --- | --- | --- | --- |
| 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |

1. **T (Toggle) Flip-Flop**:
   * **Inputs**: T (Toggle), Clock (clk)
   * **Outputs**: Q, Q'
   * **Operation**:
     + T = 0 → No change (hold).
     + T = 1 → Toggle (Q switches between 0 and 1).

**Excitation Table for T Flip-Flop**:

| **Current State (Q)** | **Next State (Q')** | **T** |
| --- | --- | --- |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

**13-Mark Question 2: Analysis and Design of Clocked Sequential Circuits**

**Question:**

Explain the design of **clocked sequential circuits** using **Moore and Mealy models**. Discuss **state minimization** and **state assignment** with examples.

**Answer:**

**Clocked Sequential Circuits:**

A clocked sequential circuit's state transitions occur on clock pulses. These circuits include memory elements like flip-flops to store the state. The behavior of the system depends on both the current state and the input values.

**Moore and Mealy Models:**

Sequential circuits are typically modeled in two ways:

1. **Moore Machine**:
   * **State Dependent Output**: In a Moore machine, the output depends only on the current state.
   * **State Diagram**: States are represented by circles, and transitions are represented by directed edges with input conditions.
   * **Example**: A 2-bit counter can be modeled using a Moore machine, where each state represents a binary number and the output corresponds to the state.
2. **Mealy Machine**:
   * **Output Dependent on Input and State**: In a Mealy machine, the output depends on both the current state and the input value.
   * **State Diagram**: Similar to Moore, but output is labeled on the transitions.
   * **Example**: A sequence detector can be modeled using a Mealy machine where the output is determined by both the current state and the input at each transition.

**State Minimization:**

State minimization is the process of reducing the number of states in a state machine without changing its behavior. This is done by combining equivalent states.

* **Equivalent States**: Two states are equivalent if, for all input sequences, they lead to the same output and subsequent states.
* **Minimization Technique**: This involves creating a state transition table and using methods like **state equivalence** to minimize the number of states.

**State Assignment:**

Once the state machine has been minimized, the states must be assigned binary codes. This process is called **state assignment**.

* **Binary Encoding**: States are given binary codes based on the number of states in the machine. The number of bits needed is determined by the equation:  
  Number of Bits≥⌈log⁡2(Number of States)⌉\text{Number of Bits} \geq \lceil \log\_2 (\text{Number of States}) \rceilNumber of Bits≥⌈log2​(Number of States)⌉
* **Example**: If there are 4 states, you would need at least 2 bits for the state encoding (00, 01, 10, 11).

**Design Example:**

Let’s design a 2-bit binary counter using the Moore model.

* **State Diagram**:
  + State S0: 00
  + State S1: 01
  + State S2: 10
  + State S3: 11
* **State Table**: Each state transition is based on the clock pulse.
  + From S0 (00) → S1 (01) on clock pulse 1.
  + From S1 (01) → S2 (10) on clock pulse 1.
  + From S2 (10) → S3 (11) on clock pulse 1.
  + From S3 (11) → S0 (00) on clock pulse 1.

**13-Mark Question 3: Triggering of Flip-Flops and Clocked Sequential Circuits**

**Question:**

Explain the **different types of triggering** for flip-flops and their significance in **clocked sequential circuits**. Also, discuss the steps involved in **designing clocked sequential circuits** and illustrate the procedure with an example.

**Answer:**

**Types of Triggering for Flip-Flops:**

The behavior of flip-flops can be controlled based on the type of triggering used. There are primarily **two types** of triggering:

1. **Level Triggering**:
   * **Definition**: A flip-flop with level triggering changes its state when the clock signal is at a particular level (either high or low).
   * **Operation**: The state changes whenever the clock is at a particular level, and the flip-flop holds that state until the clock changes again.
   * **Example**: In **positive level triggering**, the flip-flop will respond to the clock signal when it is high (logic 1).
2. **Edge Triggering**:
   * **Definition**: In **edge-triggered flip-flops**, the state changes only on the **rising** or **falling** edge of the clock signal.
   * **Operation**: The flip-flop samples its input and updates its state at the edge of the clock pulse (either when the clock signal changes from 0 to 1, called the **rising edge**, or from 1 to 0, called the **falling edge**).
   * **Example**: Most modern flip-flops, such as D, JK, or T flip-flops, are edge-triggered for better timing control.

**Significance of Triggering in Clocked Sequential Circuits:**

* **Clocked sequential circuits** rely on clock pulses to synchronize state transitions. The type of triggering affects the timing and precision of the circuit.
* **Edge triggering** is commonly used in modern circuits to ensure that state changes occur only at precise moments (either at the rising or falling edge of the clock), preventing unintended state changes during the rest of the clock cycle.

**Steps in Designing Clocked Sequential Circuits:**

1. **Define the Problem**:
   * Determine the purpose of the sequential circuit. For example, a counter, sequence detector, or shift register.
2. **Identify the States**:
   * Identify all the possible states the system can be in. Each state corresponds to a unique condition of memory elements (flip-flops).
3. **State Transition Diagram**:
   * Draw a state transition diagram showing the possible states and how the circuit transitions from one state to another based on inputs.
4. **State Table**:
   * Create a state transition table that lists the current state, input conditions, next state, and output (for Moore or Mealy models).
5. **Choose Flip-Flops**:
   * Select the appropriate type of flip-flop (SR, JK, D, T) based on the design requirements.
6. **Determine Excitation Tables**:
   * Use the excitation table to determine the required flip-flop inputs for each state transition.
7. **Implementation**:
   * Implement the circuit using the flip-flops and combinational logic to handle the inputs and outputs.

**Example: Design a 2-bit binary up counter using D flip-flops:**

1. **State Diagram**: States S0 (00), S1 (01), S2 (10), S3 (11).
2. **State Table**:

| **Current State** | **Input** | **Next State** |
| --- | --- | --- |
| 00 | 1 | 01 |
| 01 | 1 | 10 |
| 10 | 1 | 11 |
| 11 | 1 | 00 |

1. **Design Flip-Flops**: Use **D flip-flops** for each bit, and determine the D inputs based on the current state and required next state.

**13-Mark Question 4: Design of Counters and Registers in Synchronous Sequential Circuits**

**Question:**

Explain the design of **counters** and **registers** in **synchronous sequential circuits**. Discuss the working of a **4-bit up/down counter** and a **4-bit register** with an example. Also, explain how counters and registers are implemented using flip-flops.

**Answer:**

**Counters in Synchronous Sequential Circuits:**

A **counter** is a sequential circuit that generates a series of outputs based on a clock signal. Counters can be **up counters**, **down counters**, or **up/down counters**.

1. **Up Counter**: Increases its output with each clock pulse. For example, a 4-bit binary up counter counts from 0000 to 1111.
2. **Down Counter**: Decreases its output with each clock pulse. For example, a 4-bit binary down counter counts from 1111 to 0000.
3. **Up/Down Counter**: A counter that can be configured to either count up or count down based on a control input.

**4-bit Up/Down Counter Design:**

* **State Diagram**: The counter has four states: 0000, 0001, 0010, ..., up to 1111 (for the up counter), or 1111, 1110, 1101, ..., down to 0000 (for the down counter).
* **Working**: The counter increments or decrements its value based on the input control signal. If the control signal is "Up," the counter counts up; if "Down," the counter counts down.

**Design of 4-bit Up/Down Counter:**

* **Use Flip-Flops**: A 4-bit counter requires 4 flip-flops. Typically, JK flip-flops are used, as they can be configured for counting operations.
* **State Transitions**: The JK inputs are configured to toggle the flip-flops at each clock pulse, ensuring the correct count.

**State Table for 4-bit Up Counter:**

| **Current State** | **J1 K1** | **J2 K2** | **J3 K3** | **J4 K4** | **Next State** |
| --- | --- | --- | --- | --- | --- |
| 0000 | 1 | 1 | 1 | 1 | 0001 |
| 0001 | 1 | 1 | 1 | 1 | 0010 |
| ... | ... | ... | ... | ... | ... |

**Excitation Table for JK Flip-Flops:**

* Based on the current state and the required next state, use the JK excitation table to determine the necessary J and K inputs for each flip-flop.

**Registers in Synchronous Sequential Circuits:**

A **register** is a group of flip-flops used to store multi-bit data. Registers can store data, shift it, or perform other operations depending on the design.

* **4-bit Register Design**: A 4-bit register uses four flip-flops, each storing one bit of data. The D input of each flip-flop is connected to the data input, and the clock signal controls when data is stored.

**4-bit Register Example:**

* **Design**: To implement a simple **parallel load register**, the D inputs of the flip-flops are connected to the 4-bit data input. The clock pulse causes the flip-flops to store the data present on the D inputs.

**Register Operation**: When the clock signal is triggered, the data input is loaded into the register.

**Implementation of a 4-bit Shift Register:**

* A shift register is a type of register where data shifts left or right on each clock pulse.
* **Shift Operation**: In a **4-bit shift register**, each flip-flop stores a bit of data. The data in each flip-flop shifts to the next one at each clock pulse (either left or right depending on the shift control).

**Implementation of Registers and Counters Using Flip-Flops:**

* **Counter Design**: Counters like the 4-bit binary counter are designed using flip-flops and combinational logic that controls the counting operation.
* **Register Design**: Registers are created by connecting multiple flip-flops in parallel and controlling the loading of data using a clock signal.

**UNIT II: SYNCHRONOUS SEQUENTIAL LOGIC**

**Question 1:**

**Explain the analysis procedure of a clocked sequential circuit with an example.**

**Answer:**

**Analysis Procedure:**

To analyze a clocked sequential circuit:

1. **Identify Flip-Flop type** (D, JK, T, SR).
2. **Write Boolean expressions** for flip-flop inputs using the logic gates.
3. **Develop excitation table** and **present state-next state table**.
4. **Construct state table** and **state diagram**.
5. **Identify outputs** based on the type (Moore or Mealy).

**Example:**

Given a circuit with:

* **1 JK flip-flop**
* Input **X**
* Output **Q**

**J = X’Q**, **K = X**

Using JK excitation table:

| **Q(t)** | **J** | **K** | **Q(t+1)** |
| --- | --- | --- | --- |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 |

Build the **transition table**:

| **Q(t)** | **X** | **J** | **K** | **Q(t+1)** |
| --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |

Draw the **state diagram** based on Q(t) and Q(t+1).

**Question 2:**

**Compare Moore and Mealy models. Design a sequential circuit using both.**

**Answer:**

| **Feature** | **Moore Model** | **Mealy Model** |
| --- | --- | --- |
| Output Depends | Only on current state | On current state and input |
| Output Change | On clock edge | Immediate |
| Hardware | Requires more states | Fewer states |
| Timing | Synchronous | Can be asynchronous |

**Design – Sequence Detector for “101”:**

**Moore State Diagram:**

* Output is part of state.

**Mealy State Diagram:**

* Output shown on transitions.

**Moore:**  
S0 → S1 (1) → S2 (0) → S3 (1) → Output = 1 at S3  
**Mealy:**  
Output = 1 when transition from S2 to S1 with input = 1

**Conclusion:**  
Moore is simpler and stable; Mealy is faster and efficient.

**Question 3:**

**What is state minimization? Explain any one method with an example.**

**Answer:**

**State Minimization:**

It is the process of reducing the number of states in a state machine without changing its functionality.

**Partitioning Method:**

**Steps:**

1. Partition states into output-based groups.
2. Split groups based on next state transitions.
3. Repeat until no further division possible.

**Example:**

| **Present** | **Input=0** | **Input=1** | **Output** |
| --- | --- | --- | --- |
| A | B | C | 0 |
| B | A | D | 0 |
| C | D | A | 1 |
| D | C | B | 1 |

Partition by outputs:

* Group 1: A, B (output 0)
* Group 2: C, D (output 1)

Refine further by checking transitions.

Final minimized groups give fewer states with equivalent behavior.

**Question 4:**

**Design a 3-bit binary counter using T Flip-Flops. Explain its operation.**

**Answer:**

**3-bit Binary Counter:**

Counts from 000 to 111 (0 to 7)

**Flip-Flop: T type**  
T Flip-Flop toggles when T=1

**Truth Table:**

| **Count** | **Q2** | **Q1** | **Q0** |
| --- | --- | --- | --- |
| 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 |
| 2 | 0 | 1 | 0 |
| 3 | 0 | 1 | 1 |
| 4 | 1 | 0 | 0 |
| 5 | 1 | 0 | 1 |
| 6 | 1 | 1 | 0 |
| 7 | 1 | 1 | 1 |

**T Inputs:**

* **T0 = 1** (Always toggle)
* **T1 = Q0**
* **T2 = Q0 AND Q1**

**Logic Circuit:**

Connect:

* T0 = 1
* T1 = Q0
* T2 = Q0 ⋅ Q1

Use 3 T Flip-Flops and AND gates for T2 logic.

**Question 5:**

**What are registers and counters? Explain types of shift registers with neat diagrams.**

**Answer:**

**Registers:**

A register is a group of flip-flops used to store multiple bits of data.

**Types of Shift Registers:**

1. **Serial-In Serial-Out (SISO):**
   * Data enters one bit at a time.
   * Shifts right/left with clock.
2. **Serial-In Parallel-Out (SIPO):**
   * Input serially, output all bits in parallel.
3. **Parallel-In Serial-Out (PISO):**
   * Input all bits at once, shift out serially.
4. **Parallel-In Parallel-Out (PIPO):**
   * Input and output all bits together.

**Diagram:**

SISO:

IN --> [FF1] --> [FF2] --> [FF3] --> OUT

SIPO:

IN --> [FF1] --> [FF2] --> [FF3]

↓ ↓ ↓

Q1 Q2 Q3

**Counters:**

Sequential circuits that count pulses.

* **Asynchronous Counter:** Flip-flops triggered one after another.
* **Synchronous Counter:** All flip-flops triggered simultaneously.

**UNIT IV: PROCESSOR**

**13-MARK QUESTIONS AND ANSWERS**

**1. Explain the instruction execution cycle with a neat diagram.** The instruction execution cycle consists of the following steps:

1. **Fetch**: The CPU fetches the instruction from main memory using the Program Counter (PC).
2. **Decode**: The fetched instruction is decoded by the control unit to determine the operation.
3. **Execute**: The appropriate functional units perform the operation (ALU for arithmetic/logical operations).
4. **Memory Access**: If the instruction involves memory (e.g., load/store), memory is accessed.
5. **Write-back**: The result is written back to the register or memory.

A typical diagram involves components like PC, MAR, MDR, ALU, and control unit working together.

**2. Describe the building blocks of a datapath and how they work.** The datapath comprises several essential components:

* **Registers**: Temporarily store data and instructions.
* **ALU (Arithmetic Logic Unit)**: Performs arithmetic and logical operations.
* **Multiplexers**: Select data from multiple sources.
* **Buses**: Pathways for data movement.
* **Memory Units**: Store instructions and data.

These elements work under the direction of control signals to execute instructions effectively.

**3. Compare Hardwired and Microprogrammed Control Units.**

* **Hardwired Control**:
  + Uses combinational logic circuits.
  + Fast execution due to direct signal generation.
  + Difficult to modify or update.
* **Microprogrammed Control**:
  + Uses microinstructions stored in control memory.
  + Easier to modify and maintain.
  + Slightly slower due to memory fetch for control signals.

**4. Discuss data hazards in pipelining and their solutions.**

* **Data Hazards**: Occur when instructions depend on results of previous ones.
  + **Types**: RAW (Read After Write), WAR (Write After Read), WAW (Write After Write).
  + **Solutions**:
    - **Forwarding (Bypassing)**: Sends data directly between pipeline stages.
    - **Stalling**: Inserts NOPs (no operation) to wait.
    - **Instruction Reordering**: Changes instruction sequence to minimize dependency.

**5. Explain control hazards in pipelining with techniques to handle them.**

* **Control Hazards**: Arise from branch instructions affecting PC.
  + **Solutions**:
    - **Branch Prediction**: Predicts branch direction and proceeds.
    - **Delayed Branching**: Executes next instruction irrespective of branch.
    - **Pipeline Flushing**: Clears incorrect instructions .

**UNIT V: MEMORY AND I/O**

**1. Explain the memory hierarchy with examples.** Memory hierarchy organizes storage based on speed and cost:

* **Registers**: Fastest, smallest, located in CPU.
* **Cache**: Stores frequently accessed data (L1, L2, L3).
* **Main Memory (RAM)**: Volatile, used for currently executing programs.
* **Secondary Storage (HDD/SSD)**: Large, slower.
* **Tertiary Storage (CD/DVD)**: Archival use.

**2. Describe cache memory mapping techniques.**

* **Direct Mapping**: Each block maps to a specific line.
* **Associative Mapping**: Any block to any cache line.
* **Set-Associative Mapping**: Cache is divided into sets; a block maps to any line in a set.

**3. Discuss cache replacement policies.**

* **FIFO (First-In First-Out)**: Replaces oldest block.
* **LRU (Least Recently Used)**: Replaces block not used recently.
* **Random**: Random block is replaced.

LRU is most efficient but complex to implement.

**4. Explain DMA and its role in I/O operations.**

* **DMA (Direct Memory Access)** allows devices to read/write memory without CPU.
* **DMA Controller** manages transfer.
* **Benefits**:
  + Faster I/O.
  + Frees CPU for other tasks.

**5. Differentiate between Parallel and Serial I/O interfaces.**

* **Parallel I/O**:
  + Multiple bits transmitted at once.
  + Faster but uses more wires.
* **Serial I/O**:
  + One bit at a time.
  + Slower but simpler and better for long distances.