**CS3351-Digital Principles and Computer Organization**

**UNIT 1 COMBINATIONAL LOGIC**

**Question: Multiplexers, Demultiplexers, Encoders, and Decoders**

**Question:**

* Explain the design and working of **Multiplexers**, **Demultiplexers**, **Encoders**, and **Decoders** with the help of truth tables. Discuss the applications of each of these circuits in digital systems.

**Answer:**

**Multiplexers:**

A **Multiplexer (MUX)** is a combinational circuit that selects one of several inputs and passes the selected input to the output based on control lines.

**Example: 4-to-1 Multiplexer**

* **Inputs**: I₀, I₁, I₂, I₃
* **Control Lines**: S₁, S₀
* **Output**: Y

**Truth Table**:

| **S₁** | **S₀** | **Y** |
| --- | --- | --- |
| 0 | 0 | I₀ |
| 0 | 1 | I₁ |
| 1 | 0 | I₂ |
| 1 | 1 | I₃ |

**Design:**

* For a 4-to-1 MUX, use **AND gates** and **OR gates** to control which input is passed to the output.

**Demultiplexers:**

A **Demultiplexer (DEMUX)** takes a single input and channels it to one of many outputs based on control lines.

**Example: 1-to-4 Demultiplexer:**

* **Input**: D
* **Control Lines**: S₁, S₀
* **Outputs**: O₀, O₁, O₂, O₃

**Truth Table**:

| **S₁** | **S₀** | **O₀** | **O₁** | **O₂** | **O₃** |
| --- | --- | --- | --- | --- | --- |
| 0 | 0 | D | 0 | 0 | 0 |
| 0 | 1 | 0 | D | 0 | 0 |
| 1 | 0 | 0 | 0 | D | 0 |
| 1 | 1 | 0 | 0 | 0 | D |

**Design:**

* A 1-to-4 DEMUX can be implemented using **AND gates** and **NOT gates** to route the input to the appropriate output.

**Encoders:**

An **Encoder** is a combinational circuit that converts multiple input lines into a smaller number of output lines.

**Example: 4-to-2 Encoder:**

**Inputs**: D₃, D₂, D₁, D₀  
**Outputs**: A₁, A₀ (binary representation)

**Truth Table**:

| **D₃** | **D₂** | **D₁** | **D₀** | **A₁** | **A₀** |
| --- | --- | --- | --- | --- | --- |
| 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 |

**Decoders:**

A **Decoder** converts binary data into a one-hot code. It has **n inputs** and **2ⁿ outputs**.

**Example: 3-to-8 Decoder:**

**Inputs**: A₂, A₁, A₀  
**Outputs**: D₇, D₆, D₅, D₄, D₃, D₂, D₁, D₀

**Truth Table**:

| **A₂** | **A₁** | **A₀** | **D₇** | **D₆** | **D₅** | **D₄** | **D₃** | **D₂** | **D₁** | **D₀** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

Each of these combinational circuits plays a critical role in digital systems. Multiplexers and decoders help in data routing and switching, while encoders and decoders are essential for signal compression and translation.

**UNIT II: SYNCHRONOUS SEQUENTIAL LOGIC**

**1.Question:**

**Explain the design and analysis of synchronous sequential circuits using flip-flops. Discuss Moore and Mealy models, state minimization, state assignment, and circuit implementation. Also, describe the role of registers and counters in sequential logic.** *(16 Marks)*

**Answer:**

**1. Introduction to Sequential Circuits:**

Sequential circuits are digital circuits whose output depends not only on the current input but also on the past sequence of inputs. They have memory elements (like flip-flops) to store past states. They are divided into:

* **Synchronous Sequential Circuits:** Operate with a clock signal.
* **Asynchronous Sequential Circuits:** Do not use a clock.

**2. Flip-Flops (FFs):**

Flip-flops are basic memory elements used in sequential circuits.

**Types of Flip-Flops:**

* **SR Flip-Flop:** Set-Reset
* **JK Flip-Flop:** Improved SR with toggling
* **D Flip-Flop:** Data or Delay flip-flop
* **T Flip-Flop:** Toggle flip-flop

**Operation and Excitation Tables:**

| **FF Type** | **Operation Table** | **Excitation Table** |
| --- | --- | --- |
| SR | Q(next) = S + R̅Q | To go from Q to Q+ |
| JK | Toggle when J=K=1 | Determine J, K inputs for transition |
| D | Q(next) = D | D = Q(next) |
| T | Q(next) = T⊕Q | T = Q⊕Q(next) |

**3. Triggering of Flip-Flops:**

Flip-flops are triggered on the **rising edge**, **falling edge**, or **level** of the clock signal:

* **Edge-Triggered FFs** respond only during clock transitions.
* **Level-Triggered FFs** respond as long as the clock is high/low.

**4. Analysis of Clocked Sequential Circuits:**

**Steps:**

1. Identify the type of flip-flops used.
2. Derive **flip-flop input equations** using the logic diagram.
3. Construct **state table** (present state, inputs, next state, output).
4. Draw **state diagram**.

**5. Design of Clocked Sequential Circuits:**

**Steps:**

1. **Problem Statement**: Understand required behavior.
2. **Determine Number of States**.
3. **State Diagram**: Draw states and transitions.
4. **State Table**: List inputs, present states, next states, and outputs.
5. **State Assignment**: Assign binary values to states.
6. **Flip-Flop Selection**: Choose appropriate FF (D, JK, etc.).
7. **Excitation Table**: Determine input conditions for FFs.
8. **Logic Equation Derivation**: Use Karnaugh Maps to simplify.
9. **Circuit Diagram**: Draw final logic circuit.

**6. Moore and Mealy Models:**

| **Feature** | **Moore Model** | **Mealy Model** |
| --- | --- | --- |
| Output Depends | Only on present state | On present state and input |
| Complexity | Simpler, but may need more states | Fewer states, but more complex |
| Output Change | Changes on state change | Can change during transition |

**Moore Example:**

State A: Output = 0

State B: Output = 1

**Mealy Example:**

If State A and Input = 0 → Output = 0

If State A and Input = 1 → Output = 1

**7. State Minimization:**

Reduces the number of states without changing external behavior.

* **Partitioning Technique**: Group equivalent states.
* Helps in reducing hardware and cost.

**8. State Assignment:**

Assign binary values to states.

* **Binary Assignment**: Assign sequential binary codes.
* **One-hot Assignment**: Only one bit is high for each state (used in FPGAs).

**9. Circuit Implementation:**

Once logic equations are derived, implement using:

* Flip-Flops
* Combinational logic (AND, OR, NOT, etc.)
* Clock signal for synchronization

**10. Registers:**

Registers are a group of flip-flops used to store multiple bits.

* **Types**:
  + Parallel-in Parallel-out (PIPO)
  + Serial-in Serial-out (SISO)
  + Shift Registers

**Applications:**

* Temporary data storage
* Data transfer
* Counters and delay operations

**11. Counters:**

Counters are sequential circuits that go through a sequence of states on clock pulses.

**Types:**

* **Asynchronous (Ripple) Counters**
* **Synchronous Counters**
* **Up/Down Counters**
* **Modulo-n Counters**

**Applications:**

* Digital clocks
* Frequency division
* Event counting

**📝 Conclusion:**

Synchronous sequential circuits are vital in digital systems for implementing memory and control operations. With the help of flip-flops, state machines (Moore/Mealy), and components like registers and counters, designers can develop efficient digital systems that respond accurately to input sequences.

**Question 1:**

**Explain the operation of different types of Flip-Flops with their excitation tables. Also, explain the triggering methods of flip-flops.**

**Answer:**

**Flip-Flops:**

A Flip-Flop is a basic memory element in digital electronics that stores one bit of data. It has two stable states and can be used to store state information.

**1. SR Flip-Flop:**

* **Inputs:** S (Set), R (Reset)
* **Operation Table:**

| **S** | **R** | **Q(next)** | **Operation** |
| --- | --- | --- | --- |
| 0 | 0 | Q | No change |
| 0 | 1 | 0 | Reset |
| 1 | 0 | 1 | Set |
| 1 | 1 | Invalid | Invalid state |

* **Excitation Table:**

| **Q (t)** | **Q (t+1)** | **S** | **R** |
| --- | --- | --- | --- |
| 0 | 0 | 0 | X |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | X | 0 |

**2. JK Flip-Flop:**

* An improvement over SR; invalid condition is resolved.

| **J** | **K** | **Q(next)** | **Operation** |
| --- | --- | --- | --- |
| 0 | 0 | Q | No change |
| 0 | 1 | 0 | Reset |
| 1 | 0 | 1 | Set |
| 1 | 1 | Q' | Toggle |

* **Excitation Table:**

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

**3. D Flip-Flop:**

* Also called **Data or Delay Flip-Flop**. Only one input: D.

| **D** | **Q(next)** | **Operation** |
| --- | --- | --- |
| 0 | 0 | Reset |
| 1 | 1 | Set |

* **Excitation Table:**

| **Q(t)** | **Q(t+1)** | **D** |
| --- | --- | --- |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**4. T Flip-Flop:**

* Toggles on input T = 1

| **T** | **Q(next)** | **Operation** |
| --- | --- | --- |
| 0 | Q | No change |
| 1 | Q' | Toggle |

* **Excitation Table:**

| **Q(t)** | **Q(t+1)** | **T** |
| --- | --- | --- |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

**Triggering of Flip-Flops:**

Triggering refers to the transition that causes the flip-flop to change state.

* **Level Triggering:** Flip-flop responds to a specific logic level (0 or 1).
* **Edge Triggering:** Flip-flop responds to a signal transition:
  + **Positive Edge Triggering:** Responds at 0 to 1 transition.
  + **Negative Edge Triggering:** Responds at 1 to 0 transition.
* **Master-Slave Configuration:** Two flip-flops work together to avoid timing problems.

**Conclusion:** Flip-flops are essential memory elements in sequential circuits. Understanding their behavior and triggering is crucial for designing reliable digital systems.

**Question 2:**

**Design a synchronous sequential circuit using Moore and Mealy models. Explain the design steps, state diagram, state minimization, and state assignment with an example.**

**Answer:**

**Sequential Circuit Design Using Moore and Mealy Models**

**Moore Model:**

* Output depends only on the **current state**.
* Output changes with the **clock**.

**Mealy Model:**

* Output depends on **current state and input**.
* Output can change **without clock**.

**Design Steps (Common for Both Models):**

1. **Problem Statement**
2. **State Diagram**
3. **State Table**
4. **State Minimization**
5. **State Assignment**
6. **Flip-Flop Selection**
7. **Excitation Table**
8. **K-Map Simplification**
9. **Circuit Implementation**

**Example:**

**Design a sequence detector to detect "101" using a Moore and Mealy machine.**

**1. Moore Model**

**State Diagram:**

* S0 → initial
* S1 → detected 1
* S2 → detected 10
* S3 → detected 101 (Output = 1)

| **Current State** | **Input** | **Next State** | **Output** |
| --- | --- | --- | --- |
| S0 | 0 | S0 | 0 |
| S0 | 1 | S1 | 0 |
| S1 | 0 | S2 | 0 |
| S1 | 1 | S1 | 0 |
| S2 | 0 | S0 | 0 |
| S2 | 1 | S3 | 0 |
| S3 | 0 | S2 | 1 |
| S3 | 1 | S1 | 1 |

**2. Mealy Model**

| **Current State** | **Input** | **Next State** | **Output** |
| --- | --- | --- | --- |
| S0 | 0 | S0 | 0 |
| S0 | 1 | S1 | 0 |
| S1 | 0 | S2 | 0 |
| S1 | 1 | S1 | 0 |
| S2 | 0 | S0 | 0 |
| S2 | 1 | S1 | 1 |

Here, output '1' occurs immediately after input completes "101".

**State Minimization:**

* Identify equivalent states.
* Merge them if they behave identically.

**State Assignment:**

Assign binary codes:

* S0 = 00
* S1 = 01
* S2 = 10
* S3 = 11

**Flip-Flop Selection:**

Use **D Flip-Flops** for simplicity.

**K-Map Simplification:**

Create K-maps for next-state and output logic equations using state and input variables.

**Circuit Implementation:**

Draw logic diagram using flip-flops and gates as per simplified Boolean expressions.

**Conclusion:** The Moore model is safer and stable due to output change on clock edge, while the Mealy model is faster as it responds to input changes instantly. Both models are vital in sequential circuit design.

**UNIT III COMPUTER FUNDAMENTALS**

**Question 1:**

**Discuss the instruction cycle and explain how a machine instruction is executed using the fetch-decode-execute cycle.**

**Answer:**

**Instruction Cycle:**

Each instruction goes through:

1. **Fetch** – Get instruction from memory (using PC)
2. **Decode** – Identify opcode and operands
3. **Execute** – Perform operation (ALU or memory)
4. **Store** – Write back result if needed

**Fetch-Decode-Execute Cycle Diagram:**

plaintext

CopyEdit

+-----------+ +----------+ +-----------+

| FETCH | --> | DECODE | --> | EXECUTE |

+-----------+ +----------+ +-----------+

↑

UPDATE PC

**Example:**

asm

CopyEdit

MOV A, #10

ADD A, B

* Instruction fetched using PC
* Decoded into opcode and operands
* Executed using ALU
* PC is incremented

**Question 2:**

**With suitable examples, explain the interaction between high-level language and assembly language.**

**Answer:**

**High-Level Language → Assembly → Machine Code**

**High-Level Code (C):**

c

CopyEdit

int a = 5;

int b = 10;

int c = a + b;

**Equivalent Assembly:**

asm

CopyEdit

MOV R0, #5 ; a = 5

MOV R1, #10 ; b = 10

ADD R2, R0, R1 ; c = a + b

**Compilation Process:**

1. **Source Code (C/C++)**
2. **Compiler converts to Assembly**
3. **Assembler converts to Machine Code**
4. **Linker and Loader prepare executable**

**Benefits of Assembly View:**

* Optimization
* Debugging
* Understanding hardware behavior

**✅ Question 3:**

**Explain instruction encoding with formats. Show how opcodes and operands are encoded.**

**Answer:**

**Instruction Encoding:**

It is the **binary representation of machine instructions**.

**Parts of Machine Instruction:**

1. **Opcode** – Specifies operation (e.g., ADD)
2. **Operand** – Specifies data or location
3. **Addressing mode**

**Instruction Format Types:**

| **Format Type** | **Description** |
| --- | --- |
| 1-Address | 1 operand, accumulator implied |
| 2-Address | 2 operands (source, destination) |
| 3-Address | Source1, Source2, Destination |
| 0-Address | Stack-based (implied) |

**Example:**

**ADD R1, R2**

* Opcode: 0001
* R1: 001
* R2: 010

**Encoding:** 0001 001 010

**UNIT IV: PROCESSOR**

**1. Design a simple control unit using hardwired control and explain it.** A hardwired control unit uses logic gates, multiplexers, and decoders to produce control signals. Components include:

* **Instruction Register**: Holds current instruction.
* **Decoder**: Decodes opcode to identify operation.
* **Control Logic Gates**: Generate signals for data movement, ALU, etc.
* **Sequence Counter**: Determines sequence of control steps.

Advantages:

* Fast operation.
* Ideal for RISC processors. Disadvantages:
* Inflexible and hard to modify.

**2. Explain pipelining stages and discuss the hazards with solutions in detail.** Pipelining divides instruction execution into stages:

1. **IF (Instruction Fetch)**
2. **ID (Instruction Decode)**
3. **EX (Execute)**
4. **MEM (Memory Access)**
5. **WB (Write Back)**

**Hazards**:

* **Data Hazards**: Solved by forwarding, stalling.
* **Control Hazards**: Solved by prediction, flushing.
* **Structural Hazards**: Solved by duplicating hardware or scheduling.

**3. Describe in detail the architecture of a datapath with control signals for instruction execution.** The datapath includes:

* **Instruction Memory**: Stores program instructions.
* **Register File**: Holds temporary data.
* **ALU**: Performs computations.
* **Multiplexers**: Direct data paths.
* **Data Memory**: Accesses data.
* **Control Unit**: Sends control signals (e.g., RegWrite, MemRead).

Each stage in the datapath is governed by control signals for proper instruction execution.

**UNIT V MEMORY AND I/O**

**1. Explain Virtual Memory with paging and segmentation.**

* **Virtual Memory**: Enables execution of large programs by simulating more memory than physically available.
* **Paging**: Divides memory into fixed-size pages. Page tables map virtual to physical addresses.
* **Segmentation**: Divides memory into logical segments (code, data).
* **Combined**: Some systems use paged segmentation.

**2. Discuss Interrupt-driven I/O and compare it with Programmed I/O and DMA.**

* **Programmed I/O**: CPU polls devices continuously.
* **Interrupt I/O**: Device interrupts CPU when ready.
* **DMA**: Device communicates directly with memory.

**Comparison**:

* Programmed I/O: Simple, inefficient.
* Interrupt I/O: Efficient, responsive.
* DMA: Best performance, low CPU overhead.

**3. Explain USB and SATA standards in detail.**

* **USB (Universal Serial Bus)**:
  + Used for peripherals (keyboard, mouse, drives).
  + Versions: USB 1.0 (12 Mbps), 2.0 (480 Mbps), 3.0+ (up to 10 Gbps).
  + Plug-and-play, supports hot swapping.
* **SATA (Serial ATA)**:
  + Interface for connecting HDDs/SSDs.
  + Replaced older PATA.
  + Higher speeds (SATA III up to 6 Gbps).
  + Uses fewer cables, supports native command queuing.

Both provide essential communication between computer and external/internal devices.