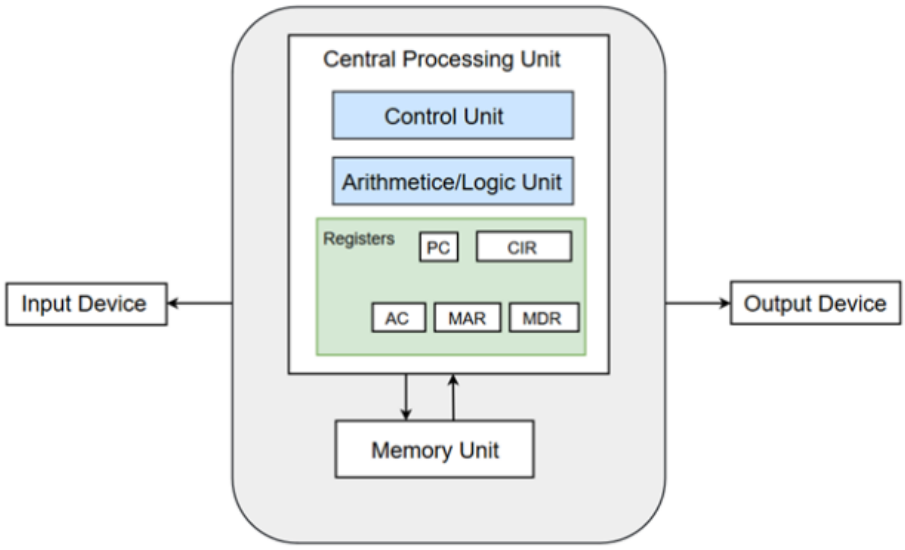
Dec 23 Sol

Q1 :

▼ a) With the help of diagram, Explain Von-Neumann's architecture.

- Von-Neumann proposed his computer architecture design in 1945 which was later known as Von-Neumann Architecture. It consisted of a Control Unit, Arithmetic, and Logical Memory
  Unit (ALU), Registers and Inputs/Outputs.
- Von Neumann architecture is based on the stored-program computer concept, where instruction data and program data are stored in the same memory. This design is still used in
  most computers produced today.
- **A Von Neumann-based computer:**
  - Uses a single processor
  - Uses one memory for both instructions and data.
  - Executes programs following the fetch-decode-execute cycle

Von-Neumann Basic Structure:



- Components of Von-Neumann Model:
  - Central Processing Unit
  - Buses
  - Memory Unit
- **Central Processing Unit :**
  - The part of the Computer that performs the bulk of data processing operations is called the Central Processing Unit and is referred to as the CPU.
  - The Central Processing Unit can also be defined as an electric circuit responsible for executing the instructions of a computer program.
  - The CPU performs a variety of functions dictated by the type of instructions that are incorporated in the computer.
  - The major components of CPU are Arithmetic and Logic Unit (ALU), Control Unit (CU) and a variety of registers.
    1. Arithmetic and Logic Unit (ALU)
       - The Arithmetic and Logic Unit (ALU) performs the required micro-operations for executing the instructions. In simple words,

- ALU allows arithmetic (add,
  subtract, etc.) and logic (AND, OR, NOT, etc.) operations to be carried out.

2. Control Unit
   - The Control Unit of a computer system controls the operations of components like ALU, memory and input/output devices.
   - The Control Unit consists of a program counter that contains the address of the instructions to be fetched and an instruction register into which instructions are fetched from memory for execution.
3. Registers
   - Registers refer to high-speed storage areas in the CPU. The data processed by the CPU are fetched from the registers.

- **Buses :**
  - Buses are the means by which information is shared between the registers in a multiple-register configuration system.
  - A bus structure consists of a set of common lines, one for each bit of a register, through which binary information is transferred one at a time.
  - Control signals determine which register is selected by the bus during each particular register transfer.
  - Von-Neumann Architecture comprised of three major bus systems for data transfer.

Bus	Description
Address Bus	Address Bus carries the address of data (but not the data) b/w the processor & the memory
Data Bus	Data Bus carries data b/w the processor, the memory unit and the input and output devices
Control Bus	Control Bus carries signals / commands from the CPU.

- **Memory Unit :**

- A memory unit is a collection of storage cells together with associated circuits needed to transfer information in and out of the storage.
- The memory stores binary information in groups of bits called words.
- The internal structure of a memory unit is specified by the number of words it contains and the number of bits in each word.
- Two major types of memories are used in computer systems:
  - RAM (Random Access Memory)
  - ROM (Read-Only Memory)

▼ b) Explain the working of S-R Flipflop

An SR flip-flop, also known as a Set-Reset flip-flop or latch, is a fundamental building block in digital electronics. It has two inputs, denoted as S (Set) and R (Reset), and two outputs, Q and $\bar{Q}$ (the complement of Q).

Here's how it works:

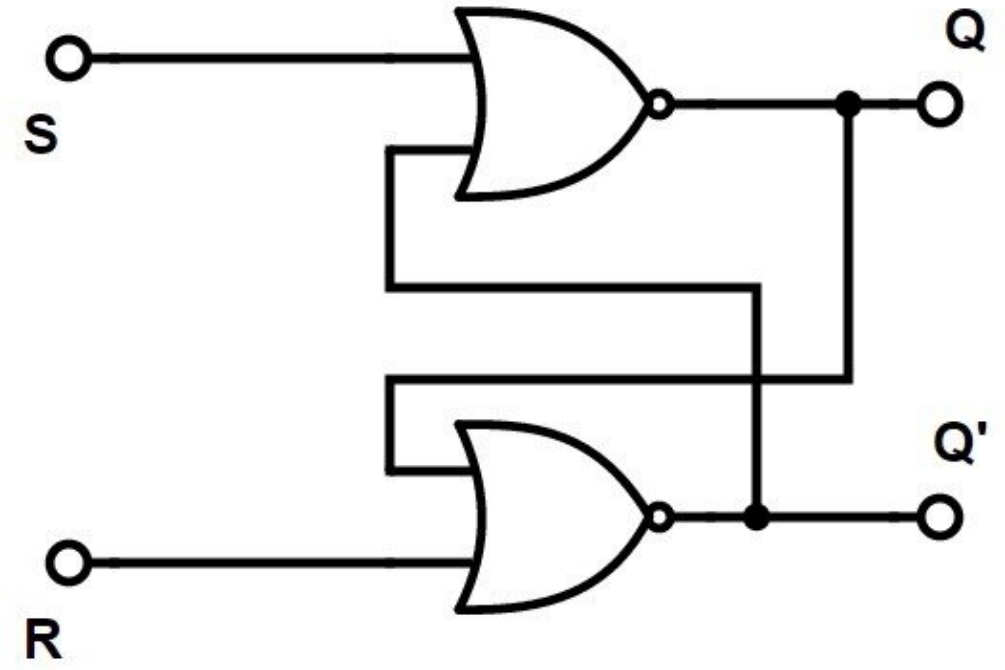
1. **State Representation**: The SR flip-flop can be in one of two stable states: SET (Q = 1, $\bar{Q}$ = 0) or RESET (Q = 0, $\bar{Q}$ = 1).
2. **Truth Table**: The behavior of an SR flip-flop is typically represented using a truth table:

S	R	Q(t)	$\bar{Q}$(t)	Description
0	0	Q(t)	$\bar{Q}$(t)	No Change (Hold)
0	1	0	1	Reset (Clear)
1	0	1	0	Set
1	1	0	0	Invalid (Undefined)

3. **Operation**:
   - When S = 0 and R = 0, the outputs retain their previous state. This condition is called the hold state.
   - When S = 0 and R = 1, the flip-flop resets. Q becomes 0 and $\bar{Q}$ becomes 1.
   - When S = 1 and R = 0, the flip-flop sets. Q becomes 1 and $\bar{Q}$ becomes 0.

- When both S and R are 1, it leads to an undefined state, so this configuration is typically avoided.

4. **NOR Gate Implementation**: An SR flip-flop can be constructed using NOR gates.



5. **Race Condition**:
    - SR flip-flops are susceptible to race conditions.
    - This occurs when both S and R inputs are activated simultaneously, causing unpredictable behavior.
    - To prevent this, additional logic or circuitry such as clocked flip-flops (like D flip-flops) are used.
6. **Applications**:
    - Memory elements in digital systems.
    - Building blocks for more complex circuits like counters, registers, and sequential logic circuits.
    - Control circuits in microprocessors and microcontrollers.

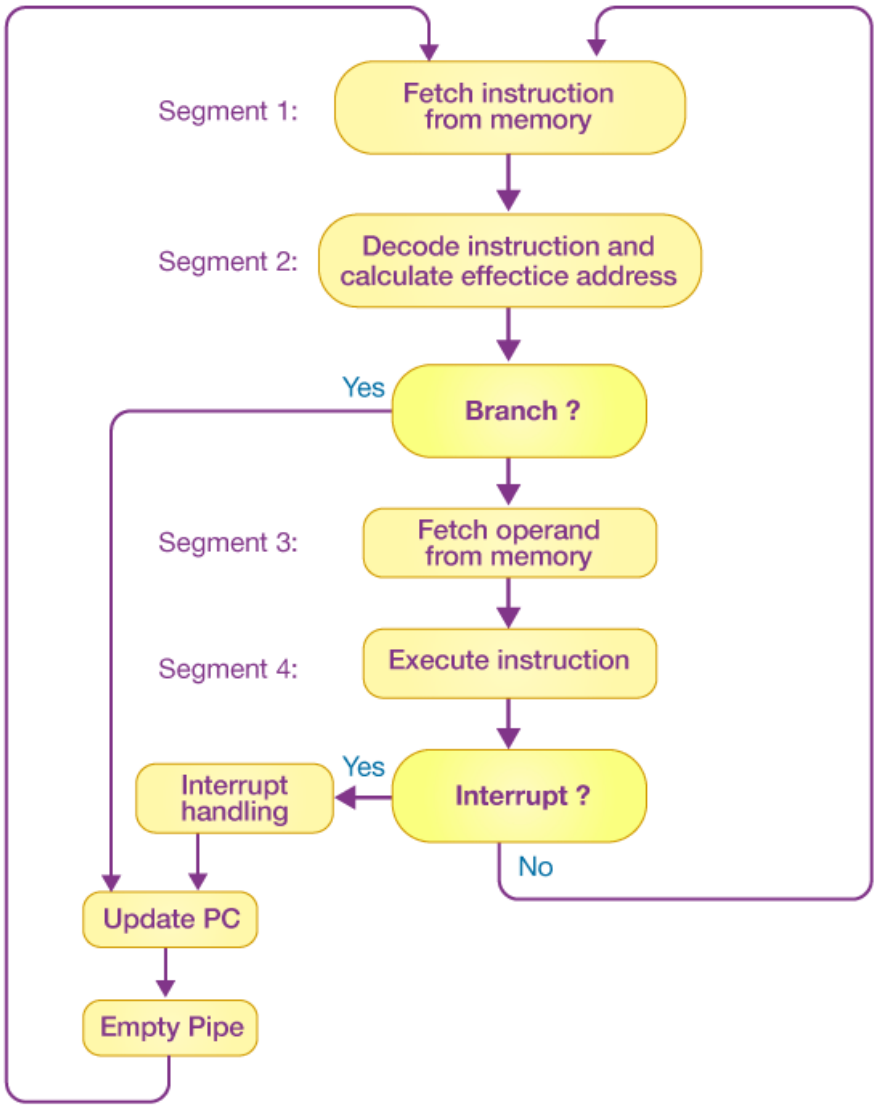
▼ d) Explain six stage instruction pipeline

A six-stage instruction pipeline is a common design in modern processors. Each stage represents a specific operation performed on an instruction as it progresses through the pipeline. Here's a breakdown of the typical stages:

1. **Instruction Fetch (IF)**:
    - Fetches the next instruction from memory.
    - The program counter (PC) is used to determine the address of the next instruction.
    - The fetched instruction is stored in an instruction register (IR) for decoding in the subsequent stages.
2. **Instruction Decode (ID)**:
    - Decodes the instruction fetched in the previous stage.
    - Determines the operation to be performed and identifies the operands.
    - Extracts any immediate values or addresses needed for the instruction.
3. **Execution (EX)**:
    - Executes the operation specified by the instruction.
    - This stage varies greatly depending on the instruction set architecture (ISA) and the type of instruction being executed.
    - Arithmetic and logic operations, memory accesses, and control flow operations are all examples of tasks performed in the execution stage.
4. **Memory Access (MEM)**:
    - If the instruction involves a memory access (e.g., load or store operation), this stage accesses the memory.
    - For a load operation, the memory access retrieves data from memory.
    - For a store operation, the memory access writes data to memory.
5. **Write Back (WB)**:
    - Writes the results of the instruction execution back to the appropriate registers.
    - For arithmetic and logic operations, the result is typically written back to a general-purpose register.

- For load operations, the data retrieved from memory is written back to a register.
- For some instructions, there might not be any data to write back (e.g., branch instructions).

6. **Pipeline Control**:
    - Manages the flow of instructions through the pipeline.
    - Handles hazards such as data hazards (e.g., when an instruction depends on the result of a previous instruction that has not yet completed) and control hazards (e.g., branch instructions that change the flow of control).



▼ e) Compare SRAM and DRAM

Feature	SRAM	DRAM
Meaning	Static RAM, retains data as long as power is on	Dynamic RAM, requires periodic refresh
Speed	Faster	Slower
Density	Lower	Higher
Power Usage	Higher	Lower
Construction	More complex	Less complex

Feature	SRAM	DRAM
Access Time	Faster access time	Slower access time
Cell Design	Uses flip-flops	Uses capacitors
Refresh	No refresh needed	Requires periodic refresh to maintain data
Usage	Cache memory, registers	Main memory in computers
Cost	More expensive	Less expensive
Applications	CPU cache, high-speed registers	Main memory in computers, graphics cards, etc.

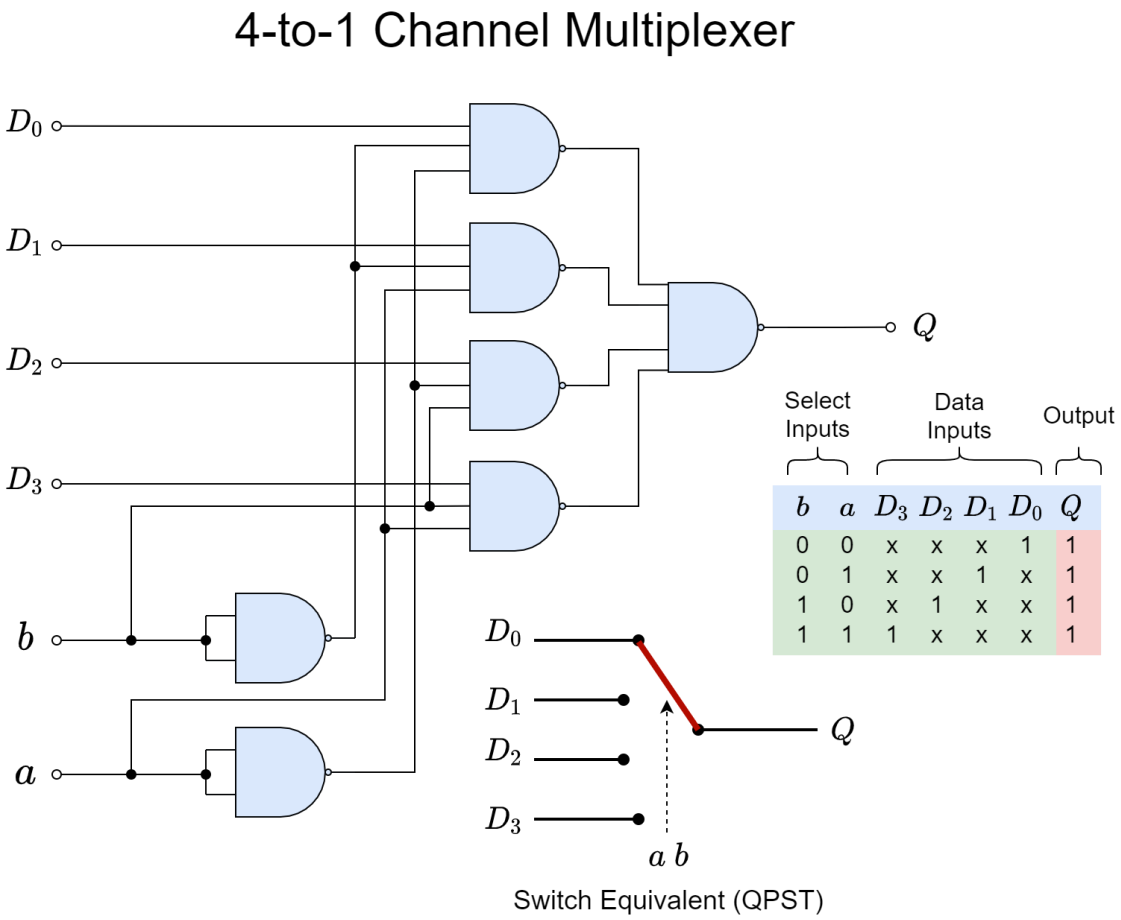
Q2 :

▼ b) Explain multiplexer and demultiplexer with a suitable diagram.

Multiplexer (MUX):

A multiplexer is a combinational circuit that selects one of many input signals and forwards it to a single output line based on the control inputs. It's like a data selector, allowing you to choose which input data to pass through.

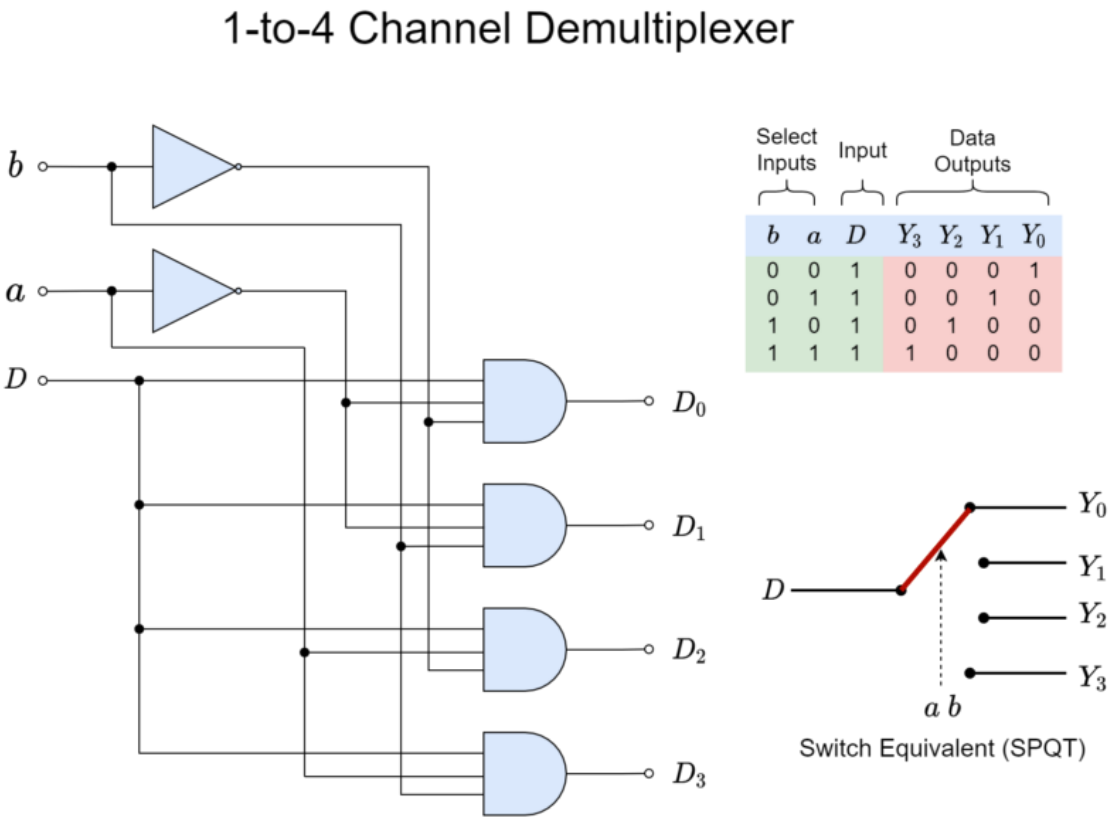
- **Inputs:** A mux has multiple data input lines (often denoted as D0, D1, D2, ..., Dn).
- **Select Lines:** It also has select lines (often denoted as S0, S1, ..., Sm) which determine which input line is routed to the output.
- **Output:** The selected input is transmitted to the output.
- **Functionality:** The selected input is determined by the binary value represented by the select lines. For example, with two select lines, you can choose between four input lines (2^2 = 4).



Demultiplexer

A demultiplexer, as the name suggests, is essentially the opposite of a multiplexer. It takes a single input and directs it to one of many possible output lines based on the control inputs.

- **Input:** A demux has one input line.
- **Select Lines:** Similar to the multiplexer, it has select lines that determine which output line receives the input.
- **Outputs:** It has multiple output lines.
- **Functionality:** The selected output line is determined by the binary value represented by the select lines.



Q4:

▼ Compare Direct and Set Associative Cache Mapping techniques in detail.

Direct Mapping:

In direct mapping, each block of main memory can only be mapped to one specific line in the cache. The mapping is determined by using a hash function or a simple modulo operation on the memory address. Here's how it works:

- **Mapping:** Each block of main memory is mapped to exactly one line in the cache.
- **Cache Structure:** The cache is divided into multiple lines, and each line can hold one block of data.
- **Associativity:** Direct-mapped caches have an associativity of 1.

- **Replacement Policy:** Typically, a least recently used (LRU) or a first-in-first-out (FIFO) replacement policy is used.
- **Advantages:**
  - Simple and easy to implement.
  - Low hardware complexity.
  - Low power consumption.
- **Disadvantages:**
  - Higher probability of cache conflicts due to limited flexibility in mapping.
  - Increased likelihood of cache misses, especially in applications with high spatial locality but poor alignment with cache line boundaries.
  - Limited parallelism in cache access due to contention for the same cache lines.

Set-Associative Mapping:

Set-associative mapping is a compromise between direct mapping and fully associative mapping. In set-associative mapping, each block of main memory can be mapped to a specific set of lines in the cache, where each set contains multiple lines. Here's how it works:

- **Mapping:** Each block of main memory can be mapped to a specific set of lines in the cache.
- **Cache Structure:** The cache is divided into multiple sets, and each set contains multiple lines.
- **Associativity:** The number of lines in each set determines the associativity of the cache (e.g., 2-way, 4-way, 8-way set-associative).
- **Replacement Policy:** Similar to direct mapping, commonly used replacement policies like LRU or FIFO are employed.
- **Advantages:**
  - Better utilization of cache space compared to direct mapping.
  - Reduced likelihood of cache conflicts due to multiple cache lines per set, providing more flexibility in mapping.

- Increased parallelism in cache access, as multiple cache lines within a set can be accessed simultaneously.

- **Disadvantages:**
  - More complex hardware compared to direct mapping, especially as associativity increases.
  - Higher power consumption and area overhead due to additional hardware required to implement set selection and replacement policies.

Comparison:

Feature	Direct Mapping	Set-Associative Mapping
Mapping	One-to-one mapping	One-to-many mapping
Flexibility	Limited flexibility in mapping	More flexibility in mapping
Cache Conflicts	More cache conflicts due to limited mapping	Reduced cache conflicts due to multiple cache lines per set
Parallelism	Limited parallelism in cache access	Increased parallelism in cache access
Hardware Complexity	Lower hardware complexity	Higher hardware complexity, especially with higher associativity
Power Consumption	Lower power consumption	Higher power consumption due to additional hardware
Performance	May lead to higher cache miss rates	Generally better performance due to reduced cache conflicts

▼ Explain various pipeline hazards with example.

Pipeline hazards are situations that arise in pipelined processors that can potentially stall or cause incorrect execution of instructions. Here are the various types of pipeline hazards:

- **Structural Hazards:**

  Structural hazards occur when hardware resources are not available for concurrent execution of instructions.

  Example:
  In a simple pipelined processor, if the instruction fetch stage, decode stage, and execute stage all require access to the memory at the same time, a

structural hazard arises. For instance, if an instruction is being fetched while another instruction is being decoded and a third instruction is being executed, all requiring memory access, the pipeline may stall because the memory cannot handle all these accesses simultaneously.

- **Data Hazards:**

  Data hazards occur when an instruction depends on the result of a previous instruction that has not yet completed.

  Example:
  Consider the following sequence of instructions:

```
ADD R1, R2, R3     ; R1 ← R2 + R3
SUB R4, R1, R5     ; R4 ← R1 - R5
```

Here, the `SUB` instruction depends on the result of the `ADD` instruction. If the `ADD` instruction has not yet completed when the `SUB` instruction enters the execute stage, a data hazard occurs. This hazard needs to be resolved to ensure correct execution.

- **Control Hazards (Branch Hazards):**

  Control hazards occur when the pipeline must make a decision on the flow of control, such as a branch instruction.

  Example:
  Consider a conditional branch instruction:

```
BEQZ R1, Label     ; Branch to Label if R1 is zero
```

The decision to branch is made after the instruction has entered the pipeline. If the condition is not known until later stages, there may be wasted work if the branch is mispredicted. For example, if the branch is predicted to be taken, but it is actually not taken, instructions fetched after the branch need to be discarded, and correct instructions need to be fetched and executed.

Q5 :

▼ Explain the characteristics of Memory.

Memory, in the context of computers, is the hardware component responsible for storing data and instructions that the CPU (Central Processing Unit) can access and manipulate. It plays a crucial role in the operation of computers by holding program instructions, data being processed, and intermediate results during computation. Memory can be broadly categorized into primary memory (main memory) and secondary memory (storage), each with its own characteristics and functionalities.

Now let's discuss various characteristics of memory:

1. **Volatility:**
    - Volatility refers to whether the memory retains its data when power is removed.
    - Primary memory (e.g., RAM) is volatile and loses its data when power is turned off, while secondary memory (e.g., HDD, SSD) is non-volatile and retains its data.
2. **Access Time:**
    - Access time is the time taken to read from or write to memory.
    - It determines how quickly the CPU can access data from memory.
    - Primary memory generally has faster access times compared to secondary memory.
3. **Capacity:**
    - Capacity refers to the amount of data that can be stored in memory.
    - It is measured in bytes, kilobytes (KB), megabytes (MB), gigabytes (GB), terabytes (TB), etc.
    - Secondary memory typically offers larger storage capacities compared to primary memory.
4. **Speed:**
    - Speed refers to how quickly data can be read from or written to memory.
    - It impacts the overall performance of the computer system.

- Faster memory speed results in quicker data access and improved system performance.

5. **Cost:**
    - Cost refers to the monetary expense associated with acquiring a certain amount of memory.
    - Different types of memory technologies have different costs per unit of storage.
    - Faster and more advanced memory technologies tend to be more expensive.
6. **Persistence:**
    - Persistence refers to whether the data stored in memory is retained even when the system is powered off.
    - Primary memory is volatile and loses its data, while secondary memory is non-volatile and retains its data.
7. **Latency:**
    - Latency is the time delay between initiating a request to memory and receiving the requested data.
    - It includes factors like access time, data transfer time, and other overheads.
    - Lower latency indicates faster memory access and improved system responsiveness.

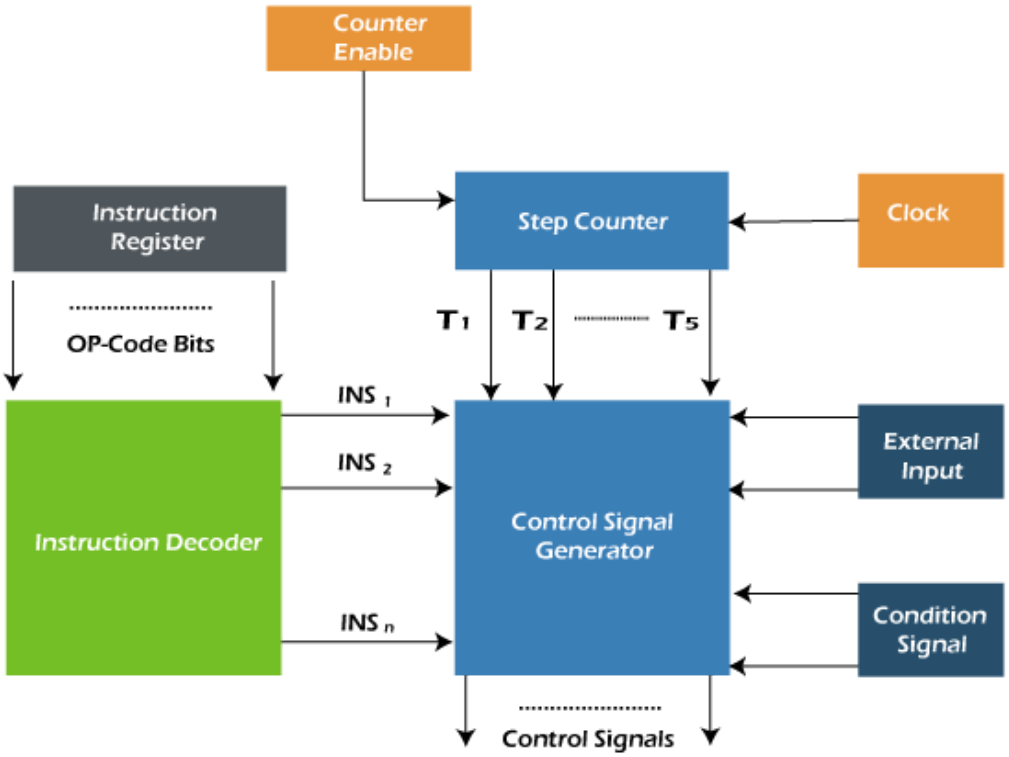
▼ Explain design of control unit with respect to microprogrammed and hardwired approach.

The control unit in a computer is responsible for coordinating and controlling the operation of various hardware components within the CPU (Central Processing Unit). It generates control signals to manage the flow of data between different parts of the CPU and between the CPU and external devices. The design of the control unit can be implemented using either a microprogrammed approach or a hardwired approach. Let's discuss both approaches:

1. Hardwired Control Unit:

In a hardwired control unit, the control logic is implemented using combinational logic circuits such as AND, OR, and NOT gates. The control signals are directly generated by the hardware based on the instruction opcode and other control inputs. Here's how the design of a hardwired control unit works:

- **Instruction Decoding:** The opcode of the instruction is decoded to determine the control signals required to execute the instruction.
- **Control Signal Generation:** The control signals are generated directly by the combinational logic circuits based on the decoded opcode and other inputs.
- **Fixed Functionality:** The functionality of the control unit is fixed and determined during the hardware design phase. Changes or updates to the control logic require modifying the hardware circuitry.
- **Efficiency:** Hardwired control units are generally more efficient in terms of speed and hardware resources since they directly generate control signals without additional overhead.
- **Example:** The design of a hardwired control unit for a simple processor might include separate circuits for instruction fetching, decoding, execution, and memory access.
- **Diagram :**



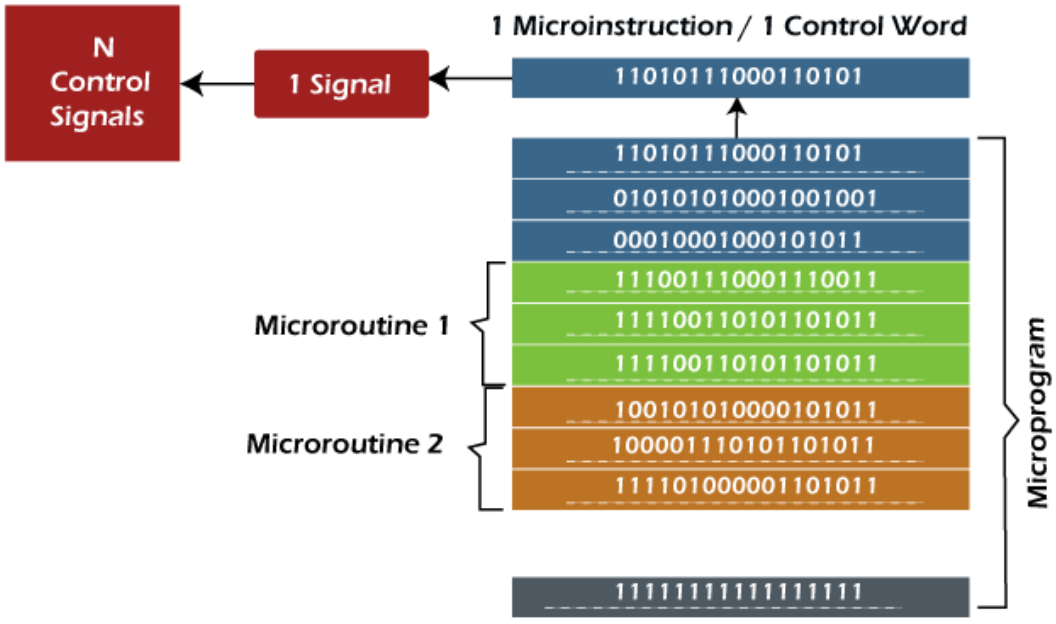
Hardwired Control Unit

2. Microprogrammed Control Unit:

In a microprogrammed control unit, the control logic is implemented using microinstructions stored in a control memory (microprogram memory). Each microinstruction corresponds to a set of control signals needed to execute a particular instruction. Here's how the design of a microprogrammed control unit works:

- **Microinstruction Execution:** The control unit fetches microinstructions from the control memory and executes them sequentially to generate the required control signals.
- **Microinstruction Format:** Each microinstruction consists of fields that specify the control signals for various components of the CPU, such as ALU operations, memory access, and register transfers.
- **Flexibility:** Microprogrammed control units are more flexible and easier to modify compared to hardwired control units. Changes or updates to the control logic can be made by modifying the microinstructions stored in the control memory.

- **Overhead:** Microprogrammed control units introduce some overhead due to the additional memory access required to fetch microinstructions.
- **Example:** The design of a microprogrammed control unit might include a control memory containing microinstructions for each instruction in the instruction set architecture (ISA).
- **Diagram :**



Q6 :

▼ Explain different addressing modes of 8086 microprocessors with example

The way for which an operand is specified for an instruction in the accumulator, in a general purpose register or in memory location, is called addressing mode.

- The 8086 microprocessors have 8 addressing modes.
- Two addressing modes have been provided for instructions which operate on register or immediate data.

- These two addressing modes are:
  - **Register Addressing:** In register addressing, the operand is placed in one of the 16-bit or 8-bit general purpose registers.
    - Example

      MOV AX, CX
      ADD AL, BL
      ADD CX, DX
  - **Immediate Addressing:** In immediate addressing, the operand is specified in the instruction itself.
    - Example
      MOV AL, 35H
      MOV BX, 0301H
      MOV [0401], 3598H
      ADD AX, 4836H
- The remaining 6 addressing modes specify the location of an operand which is placed in a memory.
- These 6 addressing modes are:
  - **Direct Addressing:** In direct addressing mode, the operand?s offset is given in the instruction as an 8-bit or 16-bit displacement element.
  - Example
    ADD AL, [0301]
    - The instruction adds the content of the offset address 0301 to AL. the operand is placed at the
      given offset (0301) within the data segment DS.
  - **Register Indirect Addressing:** The operand's offset is placed in any one of the registers BX,
    BP, SI or DI as specified in the instruction.
    - Example
      MOV AX, [BX]

    It moves the contents of memory locations addressed by the register BX to the register AX.

- **Based Addressing:** The operand's offset is the sum of an 8-bit or 16-bit displacement and the contents of the base register BX or BP. BX is used as base register for data segment, and the
  BP is used as a base register for stack segment.
  - Effective address (Offset) = [BX + 8-bit or 16-bit displacement].
  - Example
    MOV AL, [BX+05]; an example of 8-bit displacement.
    MOV AL, [BX + 1346H]; example of 16-bit displacement.
- **Indexed Addressing:** The offset of an operand is the sum of the content of an index register SI or DI and an 8-bit or 16-bit displacement.
  - Offset (Effective Address) = [SI or DI + 8-bit or 16-bit displacement]
  - Example
    MOV AX, [SI + 05]; 8-bit displacement.
    MOV AX, [SI + 1528H]; 16-bit displacement.
- **Based Indexed Addressing:** The offset of operand is the sum of the content of a base register BX or BP and an index register SI or DI.
  - Effective Address (Offset) = [BX or BP] + [SI or DI]
  - Here, BX is used for a base register for data segment, and BP is used as a base register for
    stack segment.
  - Example
    ADD AX, [BX + SI]
    MOV CX, [BX + SI]
- **Based Indexed with Displacement:** In this mode of addressing, the operand's offset is given by:
  - Effective Address (Offset) = [BX or BP] + [SI or DI] + 8-bit or 16-bit displacement
  - Example
    MOV AX, [BX + SI + 05]; 8-bit displacement
    MOV AX, [BX + SI + 1235H]; 16-bit displacement

▼ Discuss the need of DMA and explain its various techniques of data transfer

DMA (Direct Memory Access) is a feature in computer systems that allows peripherals to transfer data directly to and from memory without involving the CPU (Central Processing Unit). DMA is essential for improving overall system performance by offloading data transfer tasks from the CPU, allowing it to focus on executing application code and performing other critical tasks. Here's why DMA is needed:

1. **Offloading CPU:**
    - Without DMA, data transfer between peripherals (such as disk drives, network interfaces, and graphics cards) and memory typically involves the CPU. The CPU must initiate and manage each data transfer, which consumes its processing resources and slows down its performance.
    - DMA offloads data transfer tasks from the CPU, allowing it to focus on executing application code and performing other computational tasks.
2. **Improving Throughput:**
    - DMA enables data transfers to occur concurrently with CPU processing, improving overall system throughput and reducing the time required to complete I/O operations.
    - By transferring data directly to and from memory without CPU intervention, DMA can achieve higher data transfer rates and reduce latency.
3. **Reducing Overhead:**
    - DMA reduces the overhead associated with CPU involvement in data transfer operations. Without DMA, the CPU must handle data transfer tasks, including initiating transfers, managing buffers, and handling interrupts, which adds overhead and increases latency.
    - With DMA, these tasks are handled by dedicated DMA controllers, reducing CPU overhead and improving system efficiency.

Now, let's discuss various techniques of data transfer used in DMA:

1. **Fly-By Transfer (Burst Transfer):**
    - In fly-by transfer, DMA controller reads or writes data directly from/to memory in burst mode.
    - Data is transferred continuously without requiring the CPU to intervene for each transfer.

- Burst transfer mode improves data transfer efficiency by minimizing overhead and maximizing throughput.

2. **Block Transfer:**
    - Block transfer involves transferring a fixed-size block of data between memory and a peripheral device.
    - DMA controller transfers the entire block of data in a single operation, reducing overhead and improving efficiency.
    - Block transfer is commonly used in applications that require high-speed data transfer, such as disk I/O and network communications.
3. **Cycle Stealing:**
    - In cycle stealing, DMA controller temporarily halts CPU operation to transfer data.
    - During the data transfer, the CPU relinquishes control of the system bus to the DMA controller, allowing it to access memory directly.
    - Cycle stealing minimizes the impact on CPU performance by transferring data during CPU idle cycles.
4. **Demand Transfer:**
    - Demand transfer involves transferring data between memory and a peripheral device on-demand.
    - DMA controller initiates data transfer only when the peripheral device requests data or signals readiness to accept data.
    - Demand transfer conserves system resources by transferring data only when necessary, reducing unnecessary data transfers and conserving memory bandwidth.
5. **Memory-to-Memory Transfer:**
    - DMA controller can transfer data directly between two memory locations without CPU intervention.
    - This technique is useful for tasks such as memory copying, memory initialization, and memory clearing.