CNND DEC 2024 Answers

Q1

b) Explain the working of JK flip-flop

Working Principle:

The JK flip-flop works based on the combination of inputs (J and K) and the clock signal. The outputs Q and Q¯\overline{Q} change depending on the following conditions:

Input Conditions

1. **J = 0, K = 0 (No Change)**:
    - The output remains unchanged regardless of the clock signal.
    - Q stays the same as it was before.

2. **J = 1, K = 0 (Set)**:
    - The flip-flop sets Q to 1 (and Q bar to 0).
    - This corresponds to a "set" operation.

3. **J = 0, K = 1 (Reset)**:
    - The flip-flop resets Q to 0 (and Q bar to 1).
    - This corresponds to a "reset" operation.

4. **J = 1, K = 1 (Toggle)**:
    - The flip-flop toggles Q, meaning it switches its state to the opposite value:
        - If Q was 0, it becomes 1.
        - If Q was 1, it becomes 0.

d) Classification of memory and SRAM and DRAM differences

1. Primary Memory (Main Memory)

- Directly accessible by the CPU.
- Fast but **volatile** (data lost when power is off).
- Examples:
    - **RAM** (Random Access Memory)
        - SRAM (Static RAM)
        - DRAM (Dynamic RAM)
    - **ROM** (Read Only Memory)
        - PROM, EPROM, EEPROM

2. Secondary Memory (Storage Devices)

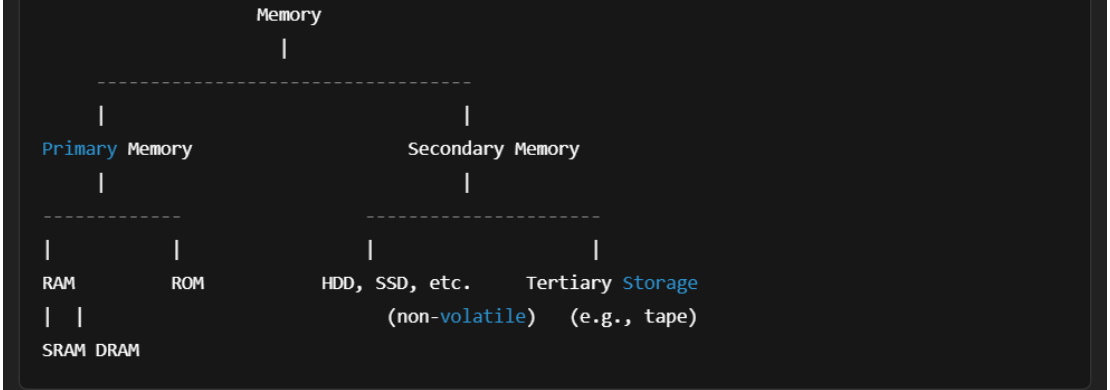
- Used for **permanent storage** of data and programs.
- **Non-volatile** (retains data without power).
- Examples:
    - HDD (Hard Disk Drive)
    - SSD (Solid State Drive)
    - CD/DVD, Pen Drives

3. Tertiary and Backup Storage

- Used for **archiving and backups**.
- Very large and cheap storage.
- Examples: Magnetic Tapes, Optical Jukeboxes.

Memory Classification Diagram

pgsql Copy Edit



Difference Between SRAM and DRAM

Feature	SRAM (Static RAM)	DRAM (Dynamic RAM)
Full Form	Static Random Access Memory	Dynamic Random Access Memory
Data Storage	Uses flip-flops (6 transistors per bit)	Uses capacitors (1 transistor + 1 capacitor)
Refresh Requirement	No refresh needed	Requires periodic refresh
Speed	Faster	Slower
Cost	Expensive	Cheaper
Density	Lower (less memory per chip)	Higher (more memory per chip)
Power Consumption	More (due to continuous power to flip-flops)	Less
Used In	Cache memory (L1, L2, L3)	Main memory (RAM modules)

e) Write a note on Amdahl's law

Amdahl's Law is a principle in computer architecture that predicts the theoretical maximum improvement in performance of a system when part of the system is enhanced. It specifically applies to parallel computing, where multiple processors work together to solve a problem.

Formula

The formula for Amdahl's Law is:

$$S = \frac{1}{(1-P) + \frac{P}{N}}$$

Where:

- **S** = Speedup (how much faster the system becomes).
- **P** = Proportion of the task that can be parallelized.
- **(1 - P)** = Proportion of the task that cannot be parallelized.
- **N** = Number of processors used.

Key Concept
Amdahl's Law emphasizes that:

1. **Sequential Bottleneck**: No matter how many processors are added, the non-parallelizable portion of the task becomes the limiting factor for performance improvement.

2. **Diminishing Returns**: As the number of processors increases, the speedup achieves diminishing returns because the sequential portion of the workload remains constant.

Example

Imagine a program where:

- 80% of the task can be parallelized (**P = 0.8**).
- 20% of the task must be executed sequentially (**1 - P = 0.2**).

If **4 processors** are used (**N = 4**):

$$S = \frac{1}{(1-0.8)+\frac{0.8}{4}} = \frac{1}{0.2+0.2} = \frac{1}{0.4} = 2.5$$

This means the program runs **2.5 times faster** with 4 processors.

Implications

- Amdahl's Law highlights the importance of reducing the sequential portion of a program to maximize performance in parallel computing.
- It sets an upper limit on the benefits of parallelization, regardless of how many processors are added.

Applications

Amdahl's Law is widely used in performance analysis, optimization, and design of parallel computing systems, helping engineers decide how many processors to use for a given task.

Q3

b) Write an assembly language program for an 8086 microprocessor to find largest 8 bit binary number.

```assembly
.model small
.stack 100h
.data
    numbers db 23h, 45h, 12h, 78h, 9Ah, 0BCh, 0DEh, 0F1h, 34h, 56h  ; Sample numbers
    largest db ?                                                    ; Stores result
.code
start:
    mov ax, @data
    mov ds, ax

    mov cx, 10              ; Counter for 10 numbers
    mov si, offset numbers  ; Pointer to array
    mov al, [si]           ; Initialize AL with first number

find_largest:
    cmp al, [si]            ; Compare current largest with next number
    jae skip_update         ; If AL >= [SI], skip update
    mov al, [si]           ; Otherwise update AL with new largest
skip_update:
    inc si                  ; Move to next number
    loop find_largest       ; Repeat until all numbers checked

    mov largest, al         ; Store final result

    mov ah, 4ch             ; Exit program
    int 21h
end start
```

Q4

b) Explain various pipeline hazards and its solutions

Pipeline hazards occur in instruction pipelining when there are conflicts that prevent the smooth execution of instructions in a pipeline. These hazards degrade performance and can result in incorrect program behavior if not addressed properly. They are classified into three types: **structural hazards**, **data hazards**, and **control hazards**. Here's a detailed explanation along with solutions:

1. Structural Hazards

- **Description:** These occur when two or more instructions require the same hardware resource simultaneously, leading to resource conflicts.
- **Example:** An instruction fetch and a data access both require access to memory, but the system doesn't have separate memory modules for these operations.
- **Solutions:**
    1. **Hardware Duplication:** Provide additional hardware resources, such as separate memory units for instruction and data access (Harvard architecture).
    2. **Stalling:** Temporarily pause the pipeline until the required resource is free. While this resolves the conflict, it introduces delays.

2. Data Hazards

- **Description:** These arise when instructions depend on the result of a previous instruction that hasn't yet been computed.

- **Types:**
    - **Read After Write (RAW):** Occurs when a subsequent instruction reads a value that hasn't been written yet.
    - **Write After Read (WAR):** Occurs when an instruction writes to a location before a previous instruction reads it.
    - **Write After Write (WAW):** Occurs when multiple instructions write to the same location in an incorrect sequence.

- **Solutions:**
    1. **Pipeline Stalling:** Introduce delays or stalls to resolve dependencies.
    2. **Compiler Optimization:** Reorder instructions during compilation to minimize dependencies and avoid hazards.

3. Control Hazards

- **Description:** These occur when the pipeline makes wrong assumptions about control flow, typically caused by branch instructions or jumps.

- **Example:** When a branch instruction is fetched, the pipeline doesn't know whether to execute the next sequential instruction or the target instruction.

- **Solutions:**
    1. **Branch Prediction:** Use hardware to predict the outcome of a branch instruction (e.g., taken or not taken). If the prediction is incorrect, the pipeline is flushed, and execution resumes with the correct instructions.
    2. **Pipeline Flushing:** Clear incorrect instructions from the pipeline after detecting a wrong branch prediction.