Computer Architecture and Organization

- Instruction types, formats, addressing modes

Imagine the CPU is a very fast, obedient worker, but it only speaks a very specific, simple language. Every command you give it is called an **Instruction**.

1. Instruction Types (The "Verbs")

Just like sentences have verbs to describe actions, instructions have types based on what they tell the CPU to do.

- **Data Transfer Instructions:** These tell the CPU to move data from one place to another.
    - *Analogy:* Moving a book from the shelf to your desk.
    - *Examples:* `MOV` (Move), `LOAD` (Load from memory), `STORE` (Save to memory), `PUSH`/`POP` (Stack operations).
- **Data Manipulation Instructions (Arithmetic & Logic):** These tell the CPU to perform math or logic checks.
    - *Analogy:* Using a calculator to add numbers or checking if a light switch is on.
    - *Examples:* `ADD`, `SUB`, `MUL`, `AND`, `OR`, `NOT`.
- **Program Control Instructions:** These tell the CPU to jump to a different part of the program (changing the flow).
    - *Analogy:* Reading a "Choose Your Own Adventure" book and skipping to page 50.
    - *Examples:* `JMP` (Jump), `CALL` (Run a function), `RET` (Return), `BZ` (Branch if Zero).
- **System Control & I/O Instructions:** These talk to external devices or control the CPU's state.
    - *Analogy:* Turning off the lights or printing a document.
    - *Examples:* `IN` (Input from keyboard), `OUT` (Output to screen), `HLT` (Halt/Stop).

2. Instruction Formats (The "Sentence Structure")

An instruction is a binary code (0s and 1s), but we can visualize it as having distinct parts. The layout of these parts is the **Instruction Format**.

Every instruction generally has two main parts:

1. **Opcode (Operation Code):** *What* to do (e.g., ADD).
2. **Operand:** *Who* to do it to (the data or the address of the data).

Computers are classified by how many "addresses" (operands) they fit in one instruction:

- **Three-Address Format:** `Opcode + Dest + Source1 + Source2`
  - *Example:* `ADD R1, A, B` (Calculates $R1 = A + B$).
  - *Pros:* Short code. *Cons:* Long binary instructions.
- **Two-Address Format:** `Opcode + Dest/Source1 + Source2`
  - *Example:* `ADD R1, B` (Calculates $R1 = R1 + B$). The destination is also one of the sources.
  - *Note:* Most common in modern computers (like Intel x86).
- **One-Address Format:** `Opcode + Source`
  - *Example:* `ADD B`.
  - *Wait, add B to what?* It assumes there is a special register called the **Accumulator** (AC). So this means $AC = AC + B$.
- **Zero-Address Format:** `Opcode` (No operands)
  - Used in **Stack Computers**. It implicitly assumes the data is on the top of the stack.
  - *Example:* `ADD` (Pops top two numbers, adds them, pushes result).

3. Addressing Modes (How to find the data)

This is often the most confusing part for beginners. Addressing modes describe **how** the CPU figures out where the operand is.

Think of it like giving a friend directions to find a specific house (the data):

- **Immediate Addressing:** The data is right there in the instruction.
  - *Analogy:* "Here is $10." (You don't have to go anywhere to get it).
  - *Code:* `MOV R1, #5` (Put the number 5 directly into Register 1).
- **Register Addressing:** The data is inside a CPU register.
  - *Analogy:* "The money is in your pocket."
  - *Code:* `MOV R1, R2` (Move data from Register 2 to Register 1).
- **Direct (Absolute) Addressing:** The instruction gives the exact memory address of the data.

    - *Analogy:* "The money is at 123 Main Street."
    - *Code:* `LOAD R1, [1000]` (Go to memory address 1000, get data, put in R1).
- **Indirect Addressing:** The instruction gives an address, but that address contains *another* address where the data actually is.
    - *Analogy:* "Go to 123 Main Street. There is a note on the door telling you the *real* address."
    - *Use case:* Pointers in programming.
- **Register Indirect Addressing:** A register holds the address of the data.
    - *Analogy:* "Check your pocket. There is a slip of paper with the address on it."
    - *Code:* `MOV AX, [BX]` (Go to the memory address stored inside register BX).
- **Relative Addressing:** The address is calculated relative to where you are right now (Instruction Pointer).
    - *Analogy:* "Walk 5 houses down from where you are standing."
    - *Use case:* Loops and `If/Else` statements (Branching).

1/10 ✕ 0 ✓ 1

1. Which part of the instruction specifies the operation to be performed (e.g., ADD, SUB)?

A. Operand

B. Opcode

✓ That's right!
Opcode stands for Operation Code. It tells the control unit what action to perform.

C. Program Counter

D. Register

2. In 8-bit 2's Complement representation, what is the binary value for -1?

A. 1000 0000

B. 1000 0001

C. 0000 0001

D. 1111 1111

✓ That's right!
In 2's complement, all 1s represent -1. (Think of rolling 0000 back one step).

2/10 ✕ 1 ✓ 1

2. In the instruction `ADD R1, #50`, which addressing mode is used for the value 50?

A. Immediate Addressing

✓ Right answer

The operand (50) is specified directly in the instruction itself. No memory access is needed.

B. Implied Addressing

C. Register Indirect

D. Direct Addressing

3/10 ✕ 1 ✓ 2

3. Which instruction format relies on a stack and typically contains no operands in the instruction itself?

A. One-Address Format

B. Two-Address Format

C. Three-Address Format

D. Zero-Address Format

✓ That's right!

Zero-address instructions (like ADD) implicitly pop the top two items from the stack, add them, and push the result.

4/10 ✕ 1 ✓ 3

4. What does the 'One-Address' instruction format implicitly use as the second operand?

A. Register R0

B. The Program Counter

C. The Accumulator (AC)

✓ That's right!

In one-address machines, `ADD X` implies `AC = AC + X`.

D. The Stack Pointer

5. In 'Register Indirect' addressing (e.g., `MOV AX, [BX]`), what is stored in the register `BX`?

A. The Opcode

B. The next instruction

C. The actual data

D. The memory address of the data

✓ That's right!

Register Indirect means the register acts as a pointer. It holds the address where the data lives in RAM.

6/10 ✕ 1 ✓ 5

6. Which addressing mode is primarily used to implement 'if' statements, loops, and branching by adding an offset to the Program Counter?

A. Direct Mode

B. PC-Relative Mode

✓ That's right!
Relative addressing adds an offset (e.g., +4, -10) to the current Program Counter to jump to a nearby location.

C. Immediate Mode

D. Stack Addressing

7. Which of the following is an example of a 'Data Transfer' instruction?

A. JMP

✕ Not quite
JMP is Program Control (Branching).

B. ADD

C. MOV (or LOAD)

✓ Right answer
MOV transfers data from source to destination without modifying it.

D. AND

8. In the instruction `LOAD R1, 1000`, if '1000' represents a memory address, which addressing mode is this?

A. Indirect Addressing

B. Direct Addressing

✓ That's right!

The instruction explicitly states the effective address (1000). The CPU goes directly to RAM address 1000.

C. Register Addressing

D. Immediate Addressing

9. Why is 'Register Addressing' (e.g., `ADD R1, R2`) faster than 'Direct Addressing' (e.g., `ADD R1, [1000]`)?

A. Register instructions have longer opcodes.

✗ Not quite

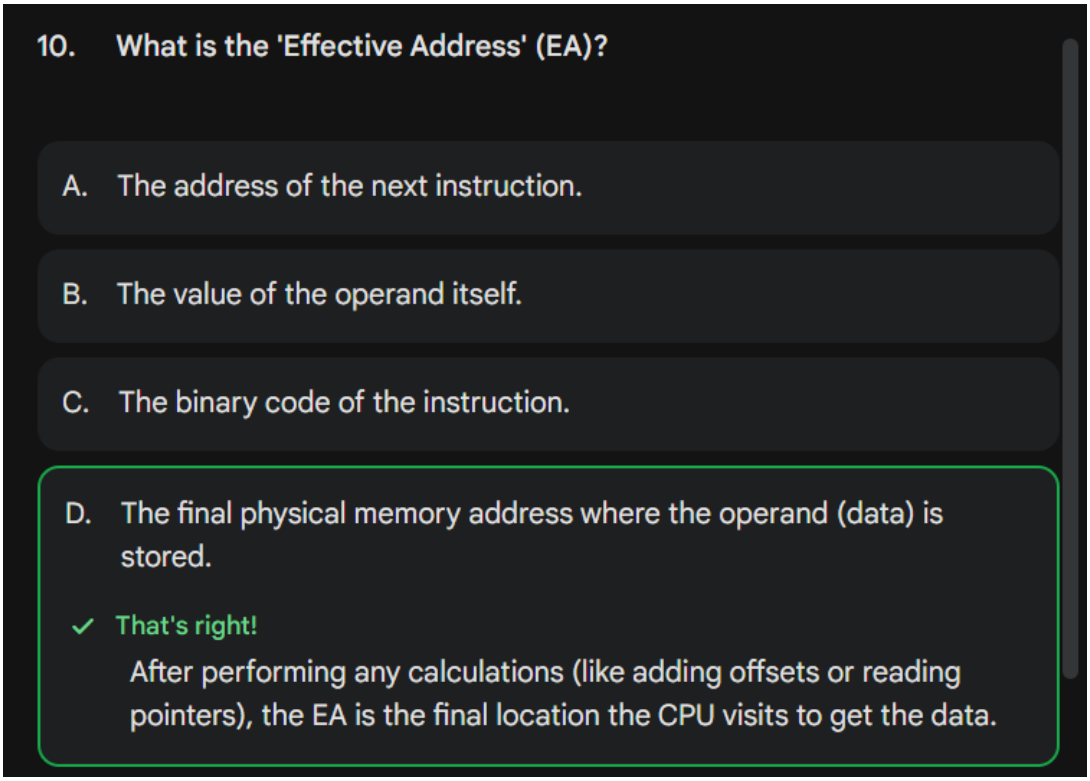
This would make them slower to fetch.

B. Registers are larger than memory.

C. Registers are inside the CPU, eliminating slow access to main memory (RAM).

✓ Right answer

Accessing RAM requires sending signals across the system bus, which is slow. Registers are physically on the CPU die and operate at CPU speed.



- Pipelining

Part 1: The Theory (Beginner's Guide)

1. What is Pipelining? (The Laundry Analogy)

Imagine you have to do 4 loads of laundry. Each load takes 4 steps:

1. **Wash** (30 mins)
2. **Dry** (30 mins)
3. **Fold** (30 mins)
4. **Put away** (30 mins)

Without Pipelining (Sequential): You put Load 1 in the washer. You wait for it to finish. You move it to the dryer. You wait. You fold. You put away. **Only then** do you start Load 2.

- *Time for 1 load:* 2 hours.
- *Time for 4 loads:* 8 hours.

With Pipelining: You put Load 1 in the washer. When it moves to the dryer, the washer is free! So, you immediately put **Load 2** into the washer. When Load 1 moves to folding, Load 2 moves to the dryer, and you put **Load 3** in the washer.

- Everyone is working at the same time.
- *Time for 4 loads:* ~3.5 hours.

In Computers: Pipelining is a technique where the CPU overlaps the execution of multiple instructions. Instead of finishing one instruction completely before starting the next, the CPU works on different parts of different instructions simultaneously.

2. The 5 Stages of a Pipeline

Most modern CPUs break the "work" of an instruction into these 5 standard stages (like the washer, dryer, folder):

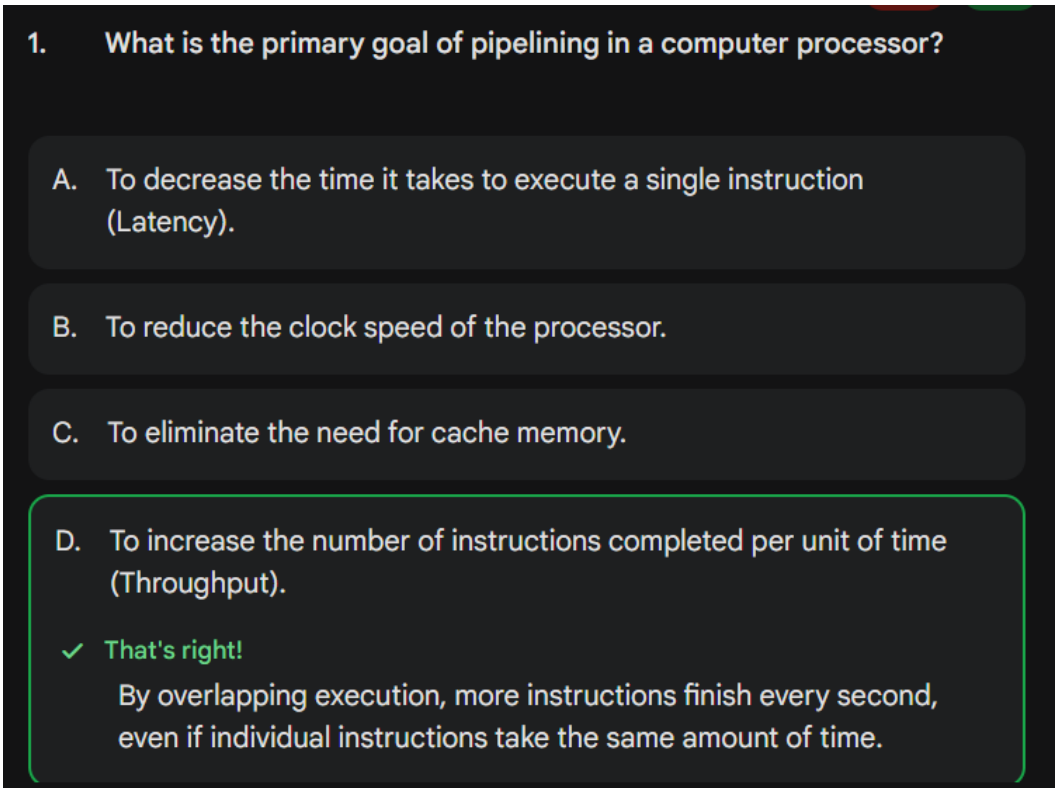
1. **IF (Instruction Fetch):** The CPU reads the instruction from memory (RAM/Cache).
2. **ID (Instruction Decode):** The CPU figures out what the instruction means (e.g., "Oh, this is an ADD command") and reads the registers needed.
3. **EX (Execute):** The ALU (Arithmetic Logic Unit) does the math or calculates the memory address.
4. **MEM (Memory Access):** If the instruction needs to read/write data to RAM (like LOAD or STORE), it happens here.
5. **WB (Write Back):** The final result is written back into the CPU register.

3. Pipeline Hazards (The Traffic Jams)

Ideally, a pipeline runs perfectly smooth (one instruction finishes every clock cycle). But sometimes, things go wrong. These problems are called **Hazards**.

- **Structural Hazards (Resource Conflict):**
  - *The Problem:* Two instructions try to use the same hardware at the same time.
  - *Analogy:* You want to put clothes in the washer, but your roommate is already using it.
- **Data Hazards (Dependency):**
  - *The Problem:* Instruction B needs a value that Instruction A hasn't calculated yet.

    - *Analogy:* You can't fold the clothes (Instruction B) until they are dry (Instruction A). You have to wait.
    - *Solution:* **Forwarding** (passing the hot clothes directly to the folder without putting them in the basket first) or **Stalling** (waiting).
- **Control Hazards (Branching):**
    - *The Problem:* The CPU has to make a choice (like an `IF` statement). It doesn't know which instruction to fetch next until the decision is made.
    - *Analogy:* You are driving and see a fork in the road. You have to stop (stall) to check the map before you know which way to turn.
    - *Solution:* **Branch Prediction** (Guessing the way! If you guess right, you save time. If wrong, you have to turn back).



3. Consider the following code:

1. `ADD R1, R2, R3` (Calculates R1)
2. `SUB R4, R1, R5` (Uses R1)

What type of hazard occurs here?

A. Structural Hazard

B. Fetch Hazard

C. Data Hazard

✓ That's right!

Instruction 2 depends on the data (R1) produced by Instruction 1. If R1 isn't ready, Instruction 2 might read an old value.

4. **How does 'Operand Forwarding' (or Bypassing) solve a Data Hazard?**

A. It guesses the value of the data.

B. It feeds the result from the ALU of one stage directly to the ALU input of the next stage.

✓ Right answer

Instead of waiting for the value to be written to a register and read back out, the CPU 'shortcuts' the data directly to where it is needed.

C. It pauses the pipeline until the data is written to the register file.

5. **A 'Control Hazard' (or Branch Hazard) is caused by which type of instruction?**

A. Logical instructions (AND, OR)

✕ Not quite

These are treated like arithmetic.

B. Branch/Jump instructions (JMP, BEQ)

✓ Right answer

These instructions change the flow of the program. The pipeline doesn't know which instruction to fetch next until the condition is checked.

C. Memory Access instructions (LOAD, STORE)

7. **If you have a 5-stage pipeline and every stage takes 1 clock cycle, how many cycles does it take to finish ONE instruction (Latency)?**

A. 1 Cycle

B. 5 Cycles

✓ That's right!

The instruction must still travel through all 5 stages (Fetch, Decode, Execute, Memory, Write).

C. 10 Cycles

D. 0.2 Cycles

8. What is 'Branch Prediction'?

A. The compiler removing all branch instructions.

B. Delaying the branch until the end of the program.

C. The CPU guessing which path a branch will take before it is actually calculated.

✓ That's right!

The CPU assumes a branch will be taken (or not) and starts fetching those instructions. If it guesses right, no time is lost.

D. Calculating the branch target address faster.

9. Ideally, what is the Speedup (S) of a pipeline with 'N' stages compared to a non-pipelined system?

A. Speedup = N - 1

B. Speedup = N

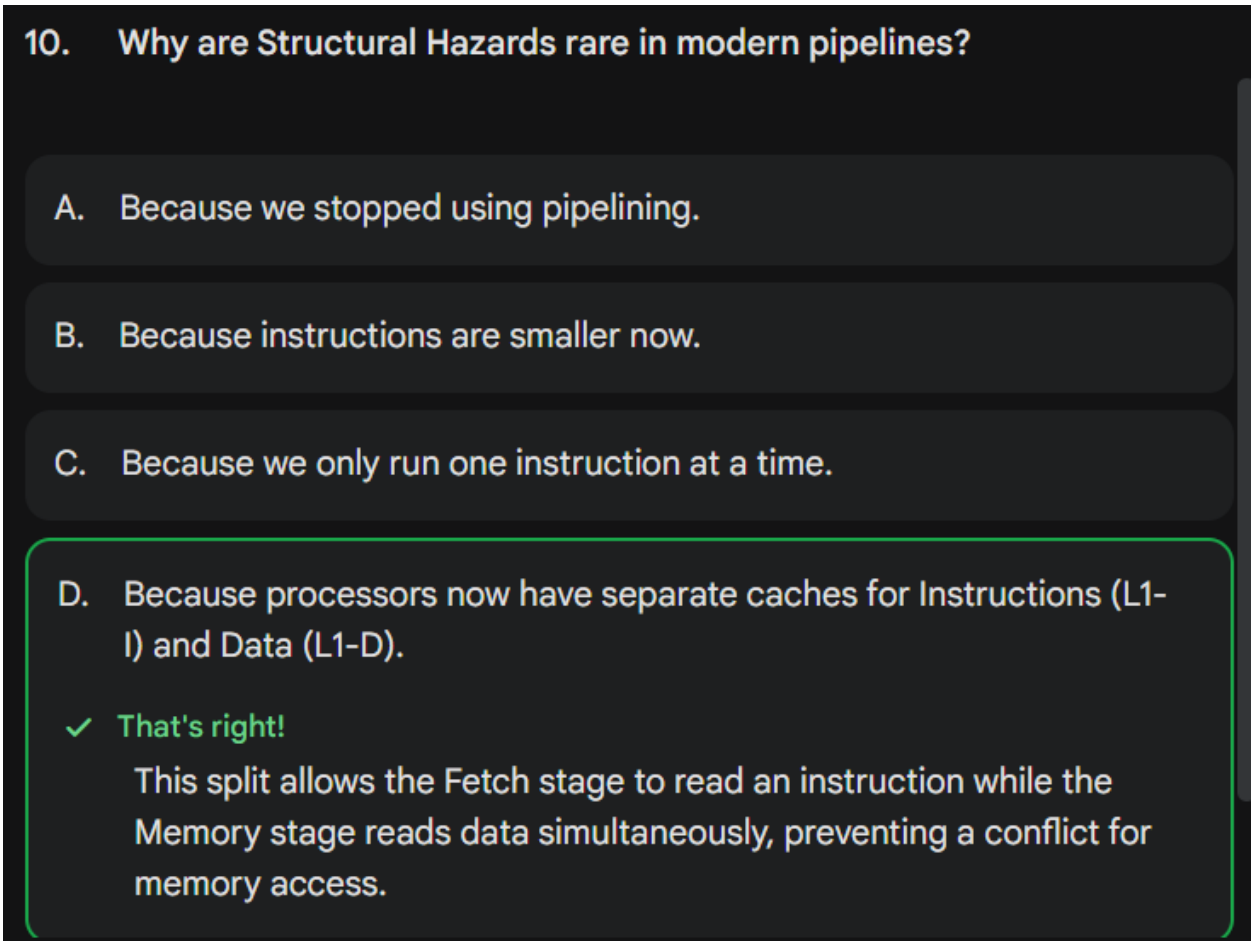
✓ Right answer

Ideally, a 5-stage pipeline is 5 times faster than a sequential processor (ignoring overhead and hazards).

C. Speedup = N^2

✕ Not quite

That would be exponential speedup, which is impossible here.



- Data representation

Part 1: The Theory (Beginner's Guide)

Computers are made of switches that can only be **ON (1)** or **OFF (0)**. This is **Binary**. But how do we turn a bunch of 0s and 1s into the complex numbers and text we see on screen?

1. Integers (Whole Numbers)

Storing positive numbers is easy (just convert to binary). But negative numbers are tricky.

- **Unsigned Integers:** Can only be positive. Used for counting things (like memory addresses).
    - *Range:* 0 to $2^n - 1$.
- **Signed Magnitude:** The first bit is the "Sign Bit" (0 = +, 1 = -). The rest is the number.
    - *Problem:* It has two zeros (+0 and -0), which confuses the computer.

- **2's Complement (The Standard):** This is the clever way modern computers store negative numbers.
    - *Analogy:* Think of an old car odometer. If it reads `0000` and you roll it *backward* one mile, it rolls over to 9999. In binary, if you have `0000` and subtract 1, it rolls back to `1111`. So, `1111` represents -1.
    - *Benefit:* It only has one zero, and math works perfectly without special rules.

2. Floating Point (Decimals / Real Numbers)

How do you store `3.14` or `0.000005` using only 0s and 1s? You use **Scientific Notation**.

- In decimal: $3.14 = 0.314 \times 10^1$.
- In binary: Computers use the **IEEE 754 Standard**.

A floating-point number is split into three parts:

1. **Sign Bit (1 bit):** Positive or Negative.
2. **Exponent (8 bits):** Moves the decimal point left or right (the "power of 2").
3. **Mantissa / Significand (23 bits):** The actual digits of the number (precision).

3. Text (Characters)

Computers assign a unique number to every letter.

- **ASCII (American Standard Code for Information Interchange):**
    - Uses 7 bits (0-127).
    - Covers English letters (A-Z, a-z), numbers, and basic symbols.
    - *Problem:* No emojis, no Chinese/Arabic/Hindi characters.
- **Unicode (The Modern Standard):**
    - Uses up to 32 bits per character.
    - Covers **every** character in every language, plus emojis! 🌎 🚀
    - **UTF-8** is the most common encoding of Unicode.

1. What is the primary disadvantage of using 'Signed Magnitude' representation for integers?

A. It uses too much memory.

B. It cannot represent negative numbers.

✕ Not quite

It handles negative numbers using a sign bit.

C. It is too slow for division.

D. It has two representations for Zero (+0 and -0).

✓ Right answer

0000 is +0 and 1000 is -0. This complicates hardware logic

2/10 ✕ 1 ✓ 1

2. In 8-bit 2's Complement representation, what is the binary value for -1?

A. 1000 0000

B. 1000 0001

C. 0000 0001

D. 1111 1111

✓ That's right!

In 2's complement, all 1s represent -1. (Think of rolling 0000 back one step).

3. Which component of the IEEE 754 Floating Point standard determines the *precision* (number of significant digits) of the number?

A. Sign Bit

B. Bias

C. Exponent

D. Mantissa (Significand)

✓ That's right!

The Mantissa holds the actual 'digits' of the number. More bits here = more precise decimal places.

4. What is the range of a signed 8-bit integer using 2's Complement?

A. 0 to 255

✗ Not quite

This is the range for an Unsigned 8-bit integer.

B. -127 to +127

C. -128 to +127

✓ Right answer

One combination is used for -128, and the extra 'negative zero' spot is used to extend the negative range by one.

5. Why do we add a 'Bias' (e.g., +127) to the exponent in IEEE 754 Floating Point?

A. To double the precision.

B. To allow exponents to be compared as unsigned integers.

✓ Right answer

It shifts the range of exponents (e.g., -126 to +127) to a positive range (1 to 254). This makes it faster for hardware to compare which number is bigger.

C. To prevent overflow.

D. To make the number positive.

6. What represents 'Infinity' in IEEE 754 Single Precision?

A. Exponent = All 1s (255), Mantissa = 0

✓ That's right!

This is the special reserved pattern for Infinity.

B. Bit 31 = 1

C. Exponent = 0, Mantissa = 0

D. Exponent = All 1s, Mantissa = Non-zero

8. In Hexadecimal (Base-16), what is the decimal equivalent of the digit 'C'?

A. 13

B. 12

✓ That's right!
The sequence is 0-9, then A(10), B(11), C(12).

C. 10

D. 11

9. What is 'Overflow' in binary arithmetic?

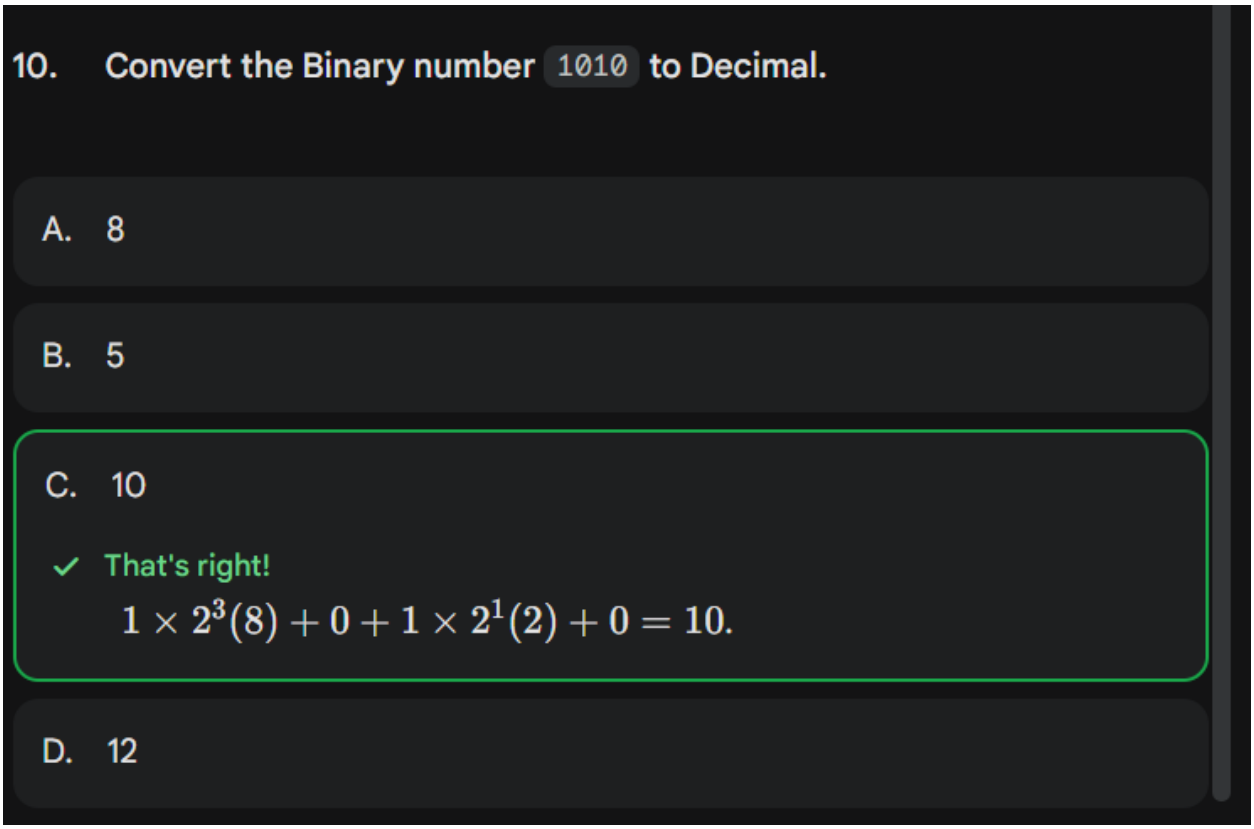
A. When the result is too large to fit in the allocated number of bits.

✓ That's right!
e.g., Adding two large 8-bit numbers results in a 9-bit number, but you only have 8 bits to store it.

B. When the sign bit changes unexpectedly.

C. When the result of an operation is 0.

D. When you try to divide by zero.



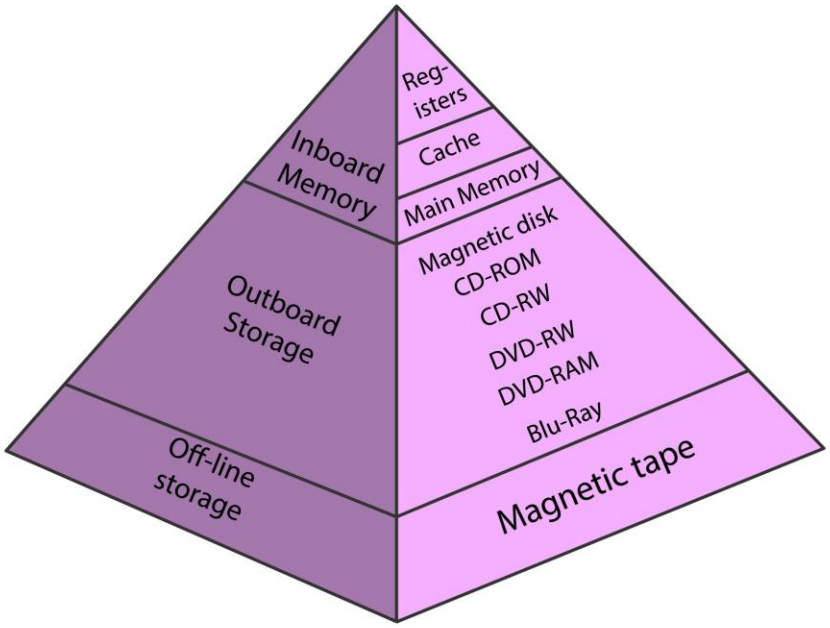
- Memory hierarchy: cache, virtual memory

Part 1: The Theory (Beginner's Guide)

Imagine you are a chef in a busy kitchen (the CPU). You need ingredients (Data) to cook (Execute Instructions).

1. The Memory Hierarchy (The Kitchen Setup)

Computers use a mix of different types of memory because we can't have memory that is huge, fast, *and* cheap all at once. It's a trade-off.



- **Registers (The Cutting Board):**
  - *Speed:* Instant.
  - *Size:* Tiny (holds just a few items).
  - *Location:* Inside the CPU.
- **Cache (The Countertop):**
  - *Speed:* Very Fast.
  - *Size:* Small.
  - *Location:* Inside the CPU. Holds ingredients you are using *right now* or will use in a second.
- **Main Memory / RAM (The Pantry):**
  - *Speed:* Moderate.
  - *Size:* Medium (GBs).
  - *Location:* On the motherboard. Holds all the ingredients for the current meal.
- **Secondary Storage / Disk (The Supermarket/Warehouse):**
  - *Speed:* Slow.
  - *Size:* Huge (TBs).
  - *Location:* Hard Drive/SSD. Holds everything you own, even if you aren't cooking it today.

2. Cache Memory (The Speed Booster)

The CPU is super fast, but RAM is relatively slow. If the CPU had to ask RAM for every single byte, it would spend most of its time waiting. **Solution:** Put a small, super-fast memory (**Cache**) between the CPU and RAM.

- **Locality of Reference (The "Rule of Thumb"):**
  - **Temporal Locality (Time):** If you used an item (data) just now, you'll probably use it again soon. (e.g., A loop counter `i`).
  - **Spatial Locality (Space):** If you used an item, you'll probably use its neighbors soon. (e.g., The next element in an array).
- **Cache Hit vs. Miss:**
  - **Hit:** The CPU finds the data in the Cache. (Fast!)
  - **Miss:** The data isn't there. The CPU has to go to slow RAM to fetch it. (Slow!)
- **Mapping (Finding the spot):**
  - *Direct Mapping:* Each block of RAM has only **one** specific parking spot in the Cache. (Simple but rigid).
  - *Associative Mapping:* A block of RAM can go **anywhere** in the Cache. (Flexible but complex to search).
  - *Set-Associative:* A compromise. A block can go into any slot within a specific "Set".

3. Virtual Memory (The Illusionist)

What if your program needs 16GB of RAM, but your computer only has 8GB? Does it crash? No. It uses **Virtual Memory**.

- **The Concept:** The computer uses a part of your Hard Drive (Disk) to pretend it is extra RAM.
- **Paging:**
  - Memory is split into fixed-size chunks called **Pages**.
  - **RAM** is split into slots called **Page Frames**.
- **The Page Table:** A map that keeps track of where every page is (Is it in RAM? Or is it sitting on the Disk?).
- **Page Fault:**
  - The CPU asks for a page.

    - The Page Table says: "Sorry, that page is currently on the Disk, not in RAM." (This is a **Page Fault**).
    - The OS pauses the program, fetches the page from the Disk, puts it into RAM (swapping out an old page if needed), and then resumes.
- **TLB (Translation Lookaside Buffer):** A tiny cache specifically for the Page Table. It speeds up looking up addresses so we don't have to read the Page Table from memory every time.

1. **Which type of memory is the fastest but also the most expensive per byte?**

A. Solid State Drive (SSD)

B. Main Memory (RAM)

C. Registers

✓ That's right!

Registers are located directly inside the CPU execution core and operate at the CPU's full clock speed.

D. Cache Memory

2. **What is the primary purpose of 'Virtual Memory'?**

A. To give the illusion of a memory space larger than the actual physical RAM.

✓ That's right!

It allows programs to address more memory than physically exists by swapping data to and from the disk.

B. To make the hard drive run faster.

C. To cache instructions for the CPU.

D. To permanently store data when the power is off.

3. Which principle explains why Cache Memory is so effective?

A. Locality of Reference

✓ Right answer

Programs tend to access the same data or nearby data repeatedly (Temporal and Spatial locality).

B. Virtualization

C. Pipelining

✕ Not quite

Pipelining is for instruction execution, not memory storage.

D. Random Access

4. What happens when the CPU requests data that is NOT present in the Cache?

A. Cache Hit

B. Page Fault

C. Interrupt

D. Cache Miss

✓ That's right!

The CPU must then fetch the data from the slower Main Memory (RAM), causing a delay.

5/10 ✕ 1 ✓ 4

5. In 'Direct Mapped Cache', where can a specific block of Main Memory be placed?

A. Anywhere in the cache.

B. In the victim buffer.

C. In exactly one specific line determined by the address.

✓ That's right!
The address modulo the cache size determines the exact, unique slot.

D. In a specific set of lines.

6. What is the function of the Translation Lookaside Buffer (TLB)?

A. To handle hard disk crashes.

B. To store the most frequently used data.

C. To store the Operating System kernel.

D. To speed up the translation of Virtual Addresses to Physical Addresses.

✓ That's right!
It caches recent Page Table entries so the CPU doesn't have to read the Page Table from RAM every time.

7. What is 'Thrashing' in the context of Virtual Memory?

A. When the hard drive is formatted.

B. When the CPU runs too fast for the memory.

C. When cache lines are overwritten randomly.

D. When the system spends more time swapping pages in and out than actually executing instructions.

✓ That's right!

This happens when the active program is larger than physical RAM, causing constant Page Faults.

8. Which Cache Write Policy updates BOTH the Cache and Main Memory simultaneously?

A. Write-Allocate

✗ Not quite

This is about what happens on a miss, not how writes are propagated.

B. Copy-Back

C. Write-Back

D. Write-Through

✓ Right answer

9. If you have a Cache Hit Ratio of 90% (0.9), what is the Miss Ratio?

A. 10

✕ Not quite

Ratios are between 0 and 1.

B. 0.9

C. 0.1

✓ Right answer

1.0 - 0.9 = 0.1 (10%).

D. 0.0

10. What is a 'Page Fault'?

A. When the printer runs out of paper.

B. When a page is written to the wrong address.

C. An error in the program code.

D. When a requested page is not in RAM and must be fetched from the Disk.

✓ That's right!

The OS interrupts, fetches the page, and updates the table.

- I/O fundamentals, techniques, DMA, interrupts

Part 1: The Theory (Beginner's Guide)

The CPU is like a Formula 1 race car driver (super fast). The Peripherals (Keyboard, Printer) are like turtles (super slow). If the CPU tried to talk to them directly, it would spend its whole life waiting.

Solution: We use an **I/O Module (Interface)** in the middle. It acts as a translator and buffer.

1. The Three Ways to Talk (I/O Techniques)

How does the CPU actually get data from a device? There are three main methods:

- **Programmed I/O (Polling):**
  - *The Concept:* The CPU keeps checking the device status register over and over again.
  - *Analogy:* You are waiting for a package. You open the front door every 10 seconds to check if the truck is there.
  - *Pros:* Simple hardware.
  - *Cons:* Wastes 99% of the CPU's time (Busy Waiting).
- **Interrupt-Driven I/O:**
  - *The Concept:* The CPU does its own work. When the device is ready, it sends a signal (Interrupt) to the CPU. The CPU stops, handles the data, and goes back to work.
  - *Analogy:* You play video games. When the doorbell rings (Interrupt), you pause the game, get the package, and resume playing.
  - *Pros:* CPU is efficient.
  - *Cons:* Not good for huge amounts of data (like transferring a 4GB movie), because the doorbell would ring millions of times!
- **Direct Memory Access (DMA):**
  - *The Concept:* The CPU hires an assistant called the **DMA Controller**. The CPU tells the DMA: "Move 1GB of data from the Disk to RAM." The CPU then goes back to other work. The DMA handles the heavy lifting and only interrupts the CPU when the *entire* job is done.
  - *Analogy:* Hiring a moving company to move your house instead of carrying every box yourself.

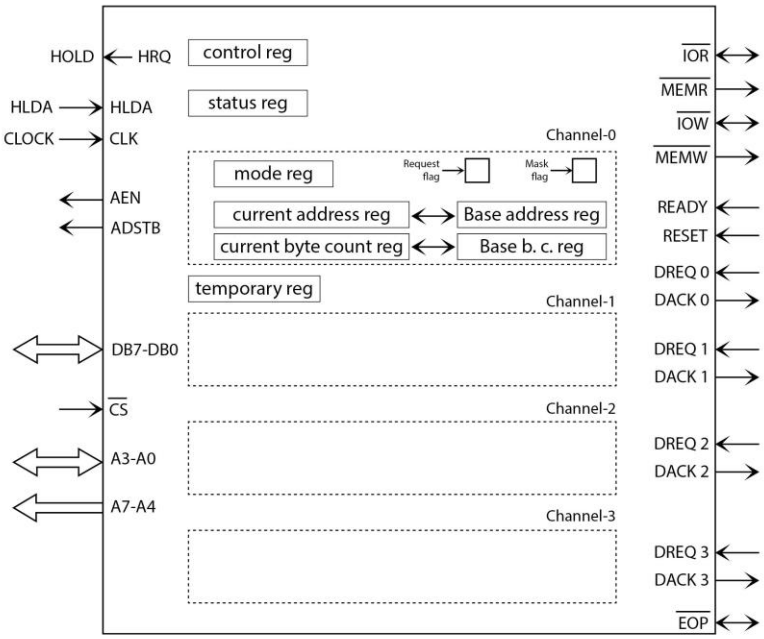


Fig: The internal block diagram of the DMA controller 8237

2. DMA Transfer Modes

How does the DMA Controller use the system bus to move data without crashing into the CPU?

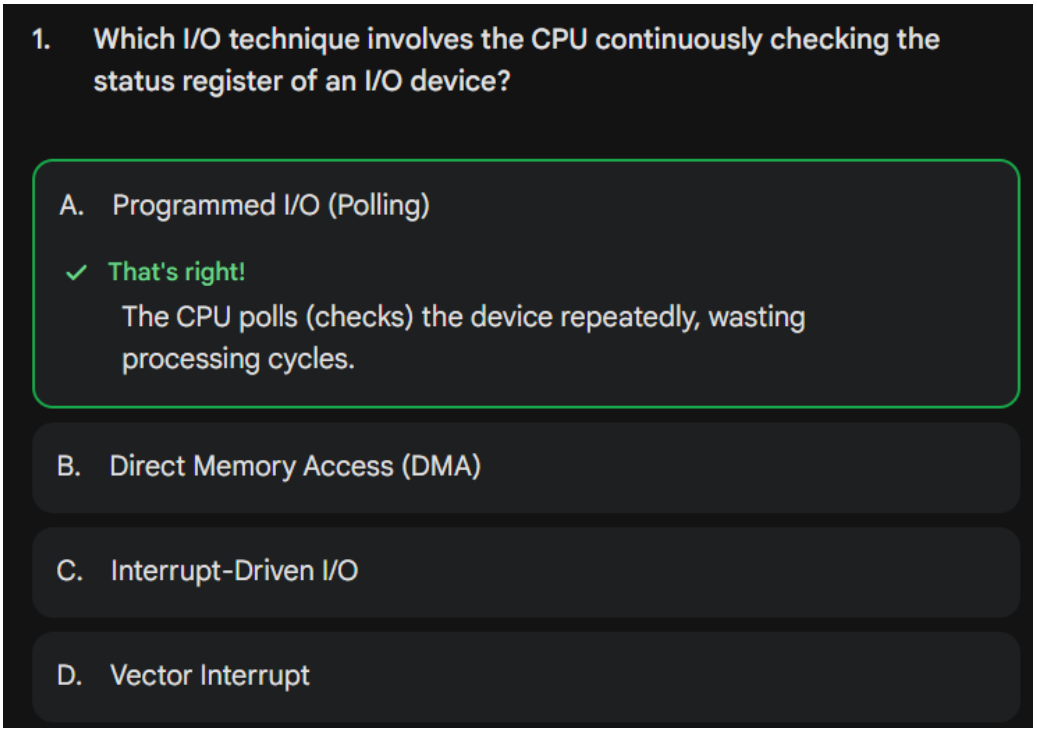
- **Burst Mode:** The DMA takes over the bus completely and transfers a huge block of data at once. The CPU has to wait (is blocked) until the burst is done.

- **Cycle Stealing:** The DMA is polite. It waits for a moment when the CPU isn't using the bus (or pauses the CPU for just *one* cycle), transfers one word of data, and then gives the bus back. It "steals" cycles here and there.

3. Interrupts (The "Hey, Listen!" Signal)

An interrupt is a signal that stops the CPU's current activity.

- **Interrupt Vector:** A number sent by the device that tells the CPU *where* to find the code to handle that specific device.
- **ISR (Interrupt Service Routine):** The actual code (mini-program) that runs when an interrupt happens (e.g., the code that reads the key you just pressed).
- **Priority:** What if the Mouse and the Fire Alarm interrupt at the same time? The CPU checks priority. (Fire Alarm > Mouse).
    - **NMI (Non-Maskable Interrupt):** An interrupt so important (like a hardware failure or power loss) that the CPU cannot ignore it.
    - **Maskable Interrupt:** An interrupt the CPU can choose to ignore (mask) if it's busy doing something more important.



2. **What is the primary advantage of DMA (Direct Memory Access) over Interrupt-Driven I/O for large data transfers?**

A. It eliminates the need for system buses.

✕ Not quite

It still uses the bus; it just controls it.

B. It reduces the CPU overhead by allowing direct transfer between I/O and Memory.

✓ Right answer

Interrupts require the CPU to stop and save context for *every byte*. DMA does it in hardware without bothering the CPU until the end.

C. It is cheaper to implement.

3. **In 'Cycle Stealing' DMA mode, what happens?**

A. The DMA controller transfers one word at a time by using a bus cycle when the CPU is not using it.

✓ That's right!

It 'steals' a cycle here and there, allowing the CPU to continue running (mostly) in parallel.

B. The CPU halts completely.

C. The DMA controller takes over the bus for the entire duration of the block transfer.

D. The memory is erased.

4. What is an 'Interrupt Vector'?

A. The priority level of the interrupt.

B. The address (or pointer) to the Interrupt Service Routine (ISR).

✓ That's right!

The vector tells the CPU *where* in memory the code to handle that specific interrupt is located.

C. The time it takes to handle an interrupt.

D. The signal wire used to send the interrupt.

6. What is the 'Interrupt Service Routine' (ISR)?

A. The delay before handling an interrupt.

B. The register that stores the interrupt status.

C. The hardware wire connecting the device.

D. The program code that runs to handle a specific interrupt.

✓ That's right!

When the mouse clicks, the ISR is the code that says 'Oh, a click! Let's move the cursor'.

7. **Which hardware configuration is used to handle multiple interrupts by chaining devices together based on priority?**

A. Daisy Chaining

✓ Right answer

The interrupt signal passes through devices in a chain. The first device in the chain has the highest priority and can block the signal from reaching others.

B. Vector Table

C. Star Topology

D. Polling Loop

8. **In 'Memory-Mapped I/O', how does the CPU communicate with I/O devices?**

A. Using standard memory instructions like `MOV` or `LOAD`.

✓ Right answer

The I/O devices are assigned addresses in the main memory space. The CPU treats them just like RAM.

B. Through the cache only.

C. Using special instructions like `IN` and `OUT`.

D. Using the ALU directly.

9. What happens to the CPU's current state (registers, PC) when an interrupt occurs?

A. It is pushed onto the Stack.

✓ That's right!
The CPU saves its 'bookmark' (Program Counter and Flags) onto the Stack so it can return exactly where it left off after the ISR finishes.

B. It is sent to the hard drive.

C. It is discarded.

D. It is moved to the DMA controller.

10. Which signal does the DMA controller send to the CPU to ask for control of the bus?

A. HOLD (Hold Request)

✓ That's right!
The DMA asserts HOLD to say 'Please pause and give me the bus'.

B. RESET

C. HLDA (Hold Acknowledge)

D. INTR

- RAID architecture

Part 1: The Theory (Beginner's Guide)

RAID stands for **Redundant Array of Independent Disks**. Ideally, we want a storage system that is **Fast**, **Huge**, and **Safe** (never loses data). A single hard drive can't be all three. **Solution:** Combine multiple cheap hard drives into one "Super Drive" team.

The Three Key Techniques:

1. **Striping (Speed):**
    a. Splitting a file into chunks and writing them to multiple drives at the same time.
    b. *Analogy:* Three people writing a book together. If they each write one chapter simultaneously, the book is finished 3x faster.
2. **Mirroring (Safety / Redundancy):**
    a. Writing the exact same data to two drives at once.
    b. *Analogy:* You write a page, and a photocopier instantly makes a copy. If you lose your page, you have the backup.
3. **Parity (Error Checking):**
    a. Using math to figure out missing data.
    b. *Analogy:* If I tell you 3 + 5 = X, you know X is 8. If I lose the 3 but keep the 5 and 8 (? + 5 = 8), you can work backward to find the missing 3.

Common RAID Levels:

- **RAID 0 (Striping):** "Zero Redundancy"
    - **Concept:** Data is split across all drives.
    - **Pros:** Super fast (Speed = Sum of all drives). Full capacity used.
    - **Cons: Zero safety.** If *one* drive fails, **all** data is lost forever.
    - *Use:* Temporary video editing cache.
- **RAID 1 (Mirroring):**
    - **Concept:** Data is duplicated on two drives.
    - **Pros:** Very safe. If one drive dies, the other takes over instantly.
    - **Cons:** Expensive. You pay for 2 drives but only get the storage of 1 (50% efficiency).
    - *Use:* Operating System drives, critical documents.
- **RAID 5 (Striping with Parity):**

    - **Concept:** Data is striped (fast), but a "Parity Block" is also saved.
    - **Pros:** Fast reads. Good capacity (N-1 drives). Can survive **1 drive failure**.
    - **Cons:** Slower writes (due to calculating parity). Rebuilding a failed drive takes a long time.
    - *Use:* General file servers.
- **RAID 6 (Double Parity):**
    - **Concept:** Like RAID 5, but with *two* parity blocks.
    - **Pros:** Can survive **2 drive failures** at once.
    - **Cons:** Slower writes. Less capacity (N-2 drives).
    - *Use:* Large archivers where safety is critical.
- **RAID 10 (RAID 1+0):**
    - **Concept:** A "Stripe of Mirrors". It combines the speed of RAID 0 with the safety of RAID 1.
    - **Pros:** Very fast and very safe.
    - **Cons:** Very expensive (only 50% capacity). Requires at least 4 drives.
    - *Use:* High-performance databases.

1. What does RAID stand for?

A. Redundant Array of Independent Disks

✓ That's right!
Originally 'Inexpensive', now 'Independent'. Ideally, it provides redundancy (backup) using multiple drives.

B. Real-time Array of Integrated Devices

C. Rapid Access of Internal Data

D. Random Array of Independent Disks

2. Which RAID level offers the highest performance but NO data redundancy (if one drive fails, everything is lost)?

A. RAID 0

✓ That's right!
RAID 0 strips data for pure speed. It has 'Zero' redundancy.

B. RAID 10

C. RAID 1

D. RAID 5

3. If you use RAID 1 (Mirroring) with two 1TB drives, what is your total usable storage capacity?

A. 1 TB

✓ That's right!
Since the second drive is an exact duplicate of the first, you only get the space of one drive.

B. 500 GB

C. 2 TB

D. 1.5 TB

4. What is the minimum number of drives required to set up RAID 5?

A. 2

✗ Not quite
2 drives are for RAID 0 or 1.

B. 5

C. 4

D. 3

✓ Right answer
You need at least 3: two for data striping and one equivalent for distributed parity.

5. **Which RAID level can survive the failure of two drives simultaneously?**

A. RAID 1

B. RAID 6

✓ That's right!
RAID 6 uses double parity blocks, allowing for two simultaneous failures.

C. RAID 0

D. RAID 5

6. **What is 'Parity' in the context of RAID?**

A. The speed of the disk rotation.

B. A method to compress data.

C. A duplicate copy of the entire drive.

D. Data used to reconstruct information if a drive fails.

✓ That's right!
Parity is a calculated value (usually XOR) used to mathematically rebuild missing data.

7. RAID 10 is a combination of which two RAID levels?

A. RAID 1 and RAID 0

✓ That's right!
It is a Stripe (RAID 0) of Mirrors (RAID 1).

B. RAID 1 and RAID 5

C. RAID 2 and RAID 3

D. RAID 5 and RAID 6

8. Which of these is a disadvantage of RAID 5 compared to RAID 10?

A. It is less common.

B. It requires more drives.

C. Slower Write Performance.

✓ That's right!
RAID 5 has to calculate parity for every write operation, which slows it down. RAID 10 just writes.

D. Lower storage capacity.

9. **Is RAID a substitute for Data Backup?**

No.

✓ Right answer

If you delete a file on RAID 1, the mirror instantly deletes it too. You still need an offline backup.

Yes, because it has redundancy.

✗ Not quite

Dangerous misconception! RAID protects against *Hardware Failure*. It does not protect against accidental deletion, viruses, fire, or theft.

10. **Which RAID level is often referred to as 'Striping with Distributed Parity'?**

A. RAID 1

B. RAID 5

✓ That's right!

RAID 5 spreads the parity blocks across all drives so no single drive is a bottleneck.

C. RAID 4

D. RAID 3

Q6. In computers, subtraction is generally carried out by

- (A) 9’s complement
- (B) 10’s complement
- (C) 1’s complement
- (D) 2’s complement
- **Correct Answer:** (D)

Step-by-Step Explanation:

Computers use **2's Complement** because it simplifies the hardware design. It allows the CPU to use the **same Adder circuit** for both addition and subtraction. To subtract $A - B$, the computer actually performs $A + (2's\ Complement\ of\ B)$.

Q10. The circuit used to store one bit of data is known as

- (A) Register
- (B) Encoder
- (C) Decoder
- (D) Flip Flop
- **Correct Answer:** (D)

Step-by-Step Explanation:

- **Flip-Flop:** The fundamental building block of sequential circuits. It is a bistable multivibrator that can hold a state of 0 or 1.
- **Register:** A group of flip-flops used to store multiple bits (e.g., a 16-bit register).
- Therefore, the basic unit for *one* bit is the Flip-Flop.

Q96. A combinational logic circuit which sends data coming from a single source to two or more separate destinations is

- (A) Decoder
- (B) Encoder
- (C) Multiplexer
- (D) Demultiplexer
- **Correct Answer:** (D)

Step-by-Step Explanation:

- **Multiplexer (MUX):** Many Inputs -> One Output (Data Selector).
- **Demultiplexer (DEMUX):** One Input -> Many Outputs (Data Distributor).
- Since the question asks for "Single source to separate destinations," it is a De-Multiplexer.

Topic 2: Computer Architecture (Instructions & Pipelining)

Q1. In Reverse Polish Notation (RPN), expression A*B+C*D is written as

- (A) AB*CD*+
- (B) A*BCD*+
- (C) AB*CD+*
- (D) A*B*CD+
- **Correct Answer:** (A)

Step-by-Step Explanation:

RPN is also known as **Postfix Notation**. We follow operator precedence (* is higher than +).

1. Group the high precedence terms: $(A * B) + (C * D)$
2. Convert groups to Postfix: $(AB*) + (CD*)$
3. Now apply the + operator to the two groups: $AB*CD*+$

Q16. The addressing mode used in an instruction of the form ADD X Y, is

- (A) Absolute
- (B) Indirect
- (C) Index
- (D) None of these
- **Correct Answer:** (C) *(Note: In some contexts, this could be interpreted differently, but Index is the standard interpretation for X Y notation in many architectures).*

Step-by-Step Explanation:

In assembly `ADD X, Y` usually implies referencing an array or a displacement.

- **Index Mode:** The effective address is calculated by adding a constant value to the contents of an index register. This is commonly used for accessing arrays (like `Array[i]`).

Q57. An instruction pipeline can be implemented by means of

- (A) LIFO buffer
- (B) FIFO buffer
- (C) Stack
- (D) None of the above
- **Correct Answer:** (B)

Step-by-Step Explanation:

A pipeline processes instructions in a specific sequence (Fetch $\rightarrow$ Decode $\rightarrow$ Execute).

- **FIFO (First-In-First-Out):** The instruction that enters the pipeline first must leave (finish) first. This queue structure ensures instructions are executed in the correct order.

Topic 3: Memory Hierarchy

Q14. The idea of cache memory is based

- (A) on the property of locality of reference
- (B) on the heuristic 90-10 rule
- (C) on the fact that references generally tend to cluster
- (D) all of the above
- **Correct Answer:** (A)

Step-by-Step Explanation:

- **Locality of Reference:** This is the core principle. It states that if a program accesses a memory location, it is likely to access the same location (Temporal) or nearby locations (Spatial) again soon. Cache stores these "likely" locations to speed up access.

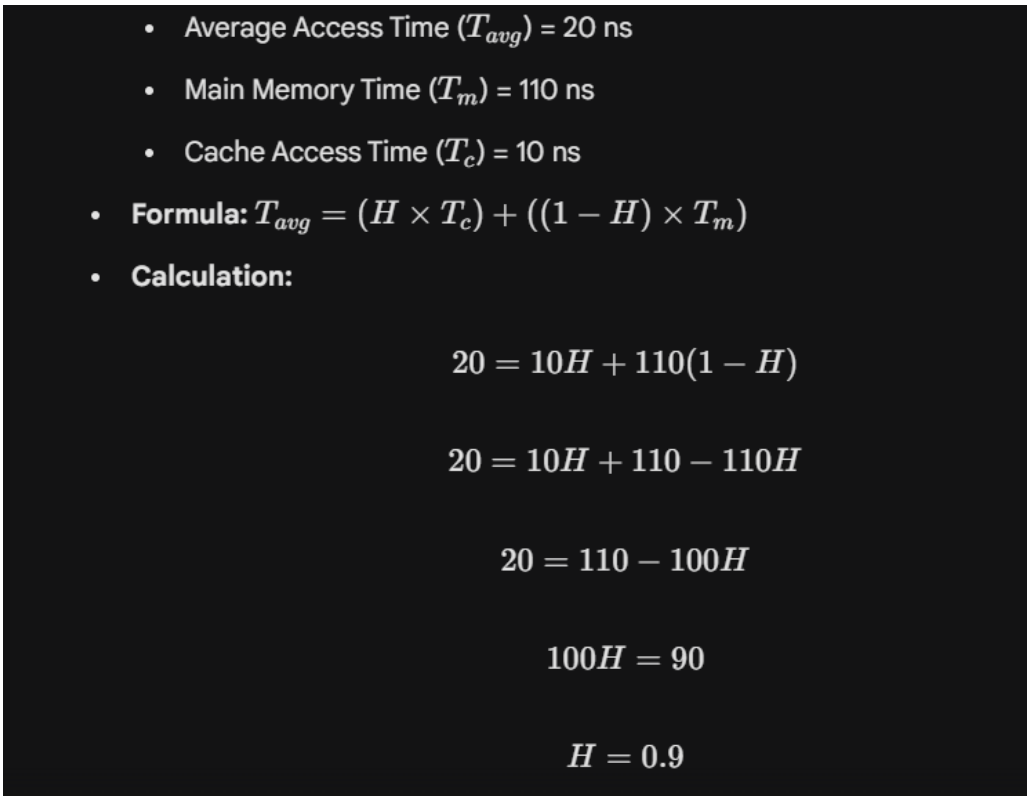
Q17. If memory access takes 20 ns with cache and 110 ns without it, then the ratio (cache uses a 10 ns memory) is

- (A) 93%
- (B) 90%

- (C) 88%
- (D) 87%
- **Correct Answer:** (B)

Step-by-Step Explanation:

This asks for the **Hit Ratio (H)**.



- **Result:** 0.9 equals **90%**.

Q23. Write Through technique is used in which memory for updating the data

- (A) Virtual memory
- (B) Main memory
- (C) Auxiliary memory
- (D) Cache memory
- **Correct Answer:** (D)

Step-by-Step Explanation:

- **Write Through:** A policy where data is written to the **Cache** and the **Main Memory** simultaneously. This ensures consistency but is slower than "Write Back".

Q102. A page fault

- (A) Occurs when there is an error in a specific page.
- (B) Occurs when a program accesses a page of main memory.
- (C) Occurs when a program accesses a page not currently in main memory.
- (D) Occurs when a program accesses a page belonging to another program.
- **Correct Answer:** (C)

Step-by-Step Explanation:

In Virtual Memory, not all pages of a program are in RAM (Main Memory) at once. If the CPU tries to access a page that is sitting on the Hard Disk but not in RAM, the hardware raises a **Page Fault** interrupt to tell the OS to fetch it.

Topic 4: I/O & Interrupts

Q19. In a vectored interrupt:

- (A) the branch address is assigned to a fixed location in memory.
- (B) the interrupting source supplies the branch information to the processor through an interrupt vector.
- (C) the branch address is obtained from a register in the processor
- (D) none of the above
- **Correct Answer:** (B)

Step-by-Step Explanation:

- **Non-Vectored:** The CPU has to poll devices to see who interrupted.
- **Vectored:** The device sends a code (Vector). The CPU uses this vector to look up the exact address of the Interrupt Service Routine (ISR) in a table.

Q76. An interface that provides I/O transfer of data directly to and form the memory unit and peripheral is termed as

- (A) DDA.

- (B) Serial interface.
- (C) BR.
- (D) DMA.
- **Correct Answer:** (D)

Step-by-Step Explanation:

DMA (Direct Memory Access): This allows peripherals to transfer data directly to/from memory without bothering the CPU for every byte. It makes the system much faster during large file transfers.

Q80. Which of the following interrupt is non maskable

- (A) INTR.
- (B) RST 7.5.
- (C) RST 6.5.
- (D) TRAP.
- **Correct Answer:** (D)

Step-by-Step Explanation:

- **Maskable:** Can be ignored/disabled by the programmer (e.g., INTR).
- **Non-Maskable (NMI):** Cannot be ignored. It is used for emergencies like power failure.
- *Note:* In the 8085 microprocessor, the NMI is named **TRAP**. In the 8086, it is simply called **NMI**. Since TRAP is the only NMI option listed here, it is the answer.