NEC Registration Examination

Computer Engineering (ACtE)

Chapter 2: Digital Logic and Microprocessor (AExE02)

Computer Engineering (ACtE)
Electronics and Communication
Telecommunication
Software Engineering
Information Technology

Complete MCQ-Focused Notes — Key Points • Full Truth Tables • K-Map Guide • Diagrams • Practice MCQs

COMPLETE NOTES WITH SOLVED MCQ AND PYQS

EASY EXPLANATION



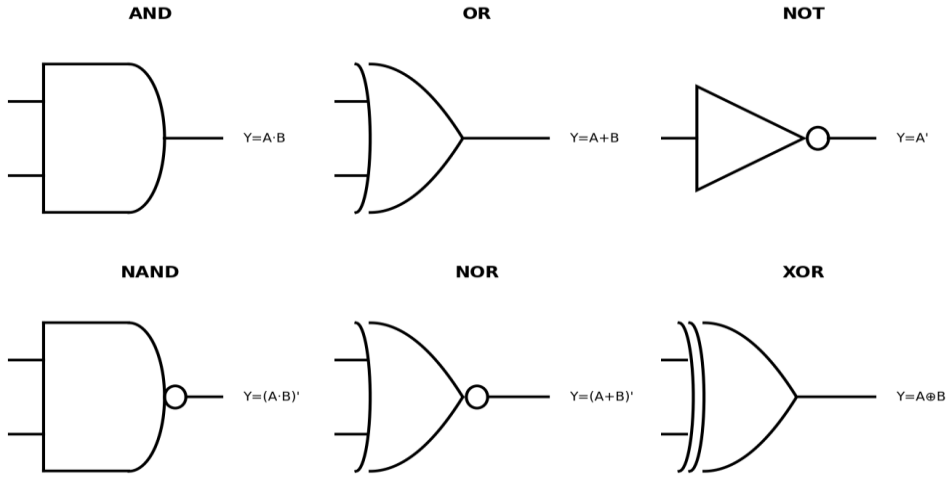
2.1 Digital Logic

Number Systems

System	Base	Digits used
Binary	2	0, 1
Octal	8	0–7
Decimal	10	0–9
Hexadecimal	16	0–9, A–F

- **1's complement:** invert all bits. **2's complement:** 1's complement + 1 → used for signed number representation & subtraction.

Logic Gates — Symbols & Boolean Expressions



- Full truth tables for all basic 2-input gates:

A	B	AND (A·B)	OR (A+B)	NAND	NOR	XOR (A⊕B)	XNOR
0	0	0	0	1	1	0	1
0	1	0	1	1	0	1	0
1	0	0	1	1	0	1	0
1	1	1	1	0	0	0	1

A	NOT (A')
0	1
1	0

- **XOR:** output is 1 only when inputs **differ** (odd number of 1s) — used as a bit-comparator/parity generator.
- **XNOR:** output is 1 only when inputs are **the same** — used as an equality checker.

A + A' = 1 A · A' = 0

A + AB = A A(A+B) = A (Absorption)

De Morgan's: (A+B)' = A'·B' (A·B)' = A'+B'

Easy Explanation

NAND and NOR are called 'Universal Gates' — any logic function (AND, OR, NOT) can be built using ONLY NAND gates, or ONLY NOR gates. This is a very common MCQ fact.

SOP & POS

- **SOP (Sum of Products):** OR of AND terms, built from **minterms** (rows where output = 1). Example: Y = A'B + AB'.
- **POS (Product of Sums):** AND of OR terms, built from **maxterms** (rows where output = 0).

Karnaugh Map (K-Map) — Full Guide

- A **graphical method** to simplify Boolean expressions (minimize gates, cost, and delay) by grouping adjacent 1's (SOP) or 0's (POS) — based on **Gray code adjacency** (neighboring cells differ by only ONE bit).

Variables (n)	Map Size	Variables per Axis	Total Cells
2	2×2	1, 1	4
3	2×4	1, 2	8
4	4×4	2, 2	16
5	4×8	2, 3	32
6	8×8	3, 3	64

4-Variable K-Map (AB \ CD)

AB \ CD	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

K-Map Grouping Rules

Group Size (cells)	Variables Eliminated
1 cell	0 (no simplification — full minterm)
2 cells	1 variable eliminated
4 cells	2 variables eliminated
8 cells	3 variables eliminated
16 cells	4 variables eliminated

- Rules: group sizes must be **powers of 2** (1,2,4,8,16...); groups must be **rectangular/contiguous**; groups **can wrap around edges** (top-bottom, left-right); **overlapping groups are allowed**; always pick the **largest possible group** containing a given 1.

- **Don't cares (X):** can be treated as **either 0 or 1**, whichever helps form a **larger group** — used when certain input combinations never occur in practice.

SOP: group the 1's → OR the resulting product terms

POS: group the 0's → AND the resulting sum terms

Easy Explanation

K-map cell order along each axis is Gray code (00,01,11,10) — NOT normal binary (00,01,10,11) — so that only ONE bit changes between neighbouring cells, which is exactly what makes adjacent-cell grouping valid.

Probable MCQs: Digital Logic

1. Which gates are known as 'Universal Gates'?

a) AND, OR
b) NAND, NOR
c) XOR, XNOR
d) NOT only
✓Answer: b) NAND, NOR
Why: NAND and NOR alone can implement any Boolean function, hence 'universal'.

2. According to De Morgan's theorem, (A+B)' equals:

a) A'+B'
b) A'·B'
c) A·B
d) A+B
✓Answer: b) A'·B'
Why: De Morgan's first theorem: (A+B)' = A'·B'.

3. In a K-map, adjacent cells are arranged so that:

a) Two bits change
b) Only one bit changes (Gray code)
c) All bits change
d) No pattern is followed
✓Answer: b) Only one bit changes (Gray code)
Why: Gray code ordering ensures only 1 bit differs between adjacent cells, enabling grouping.

4. A group of 8 cells in a K-map eliminates how many variables?

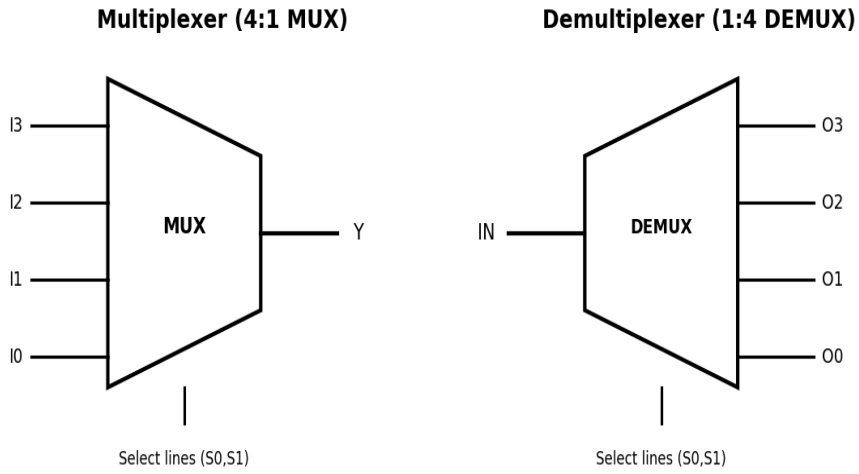
a) 1
b) 2
c) 3
d) 4
✓Answer: c) 3
Why: Group size 2^k eliminates k variables; 8 = 2^3 eliminates 3 variables.

5. The XOR gate produces an output of 1 when:

a) Both inputs are 0
b) Both inputs are 1
c) The inputs differ
d) Always, regardless of inputs
✓Answer: c) The inputs differ
Why: XOR (exclusive OR) outputs 1 only when its inputs are different.

2.2 Combinational and Arithmetic Circuits

Multiplexer & Demultiplexer



S1	S0	MUX Output
0	0	I0
0	1	I1
1	0	I2
1	1	I3

MUX: 2^n inputs need n select lines (e.g. 4:1 MUX → 2 select lines)

DEMUX: 1 input, 2^n outputs, n select lines

- **MUX** = data selector (many → one); **DEMUX** = data distributor (one → many).

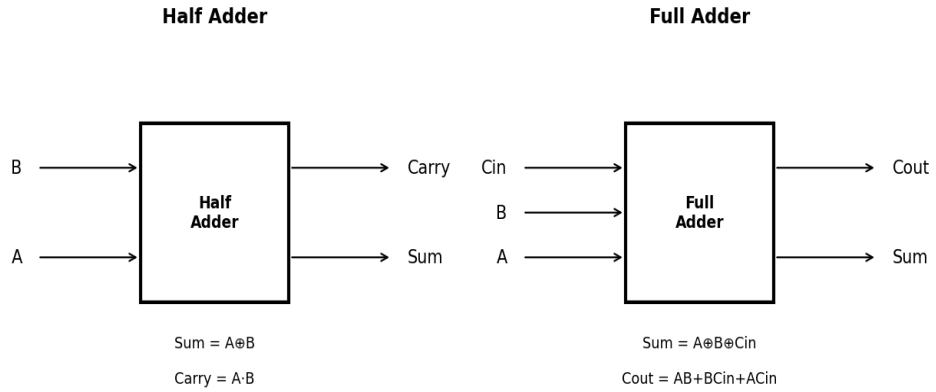
Encoder & Decoder

- **Encoder:** converts **2^n input lines → n output lines** (e.g. 8-to-3 priority encoder — opposite of decoder).
- **Decoder:** converts **n input lines → 2^n output lines** (e.g. 3-to-8 decoder — only one output active at a time). Used in memory address decoding.

Easy Explanation

Simple memory trick: 'En-CODE-r' compresses/codes many lines into few (like zipping); 'DE-code-r' un-zips few lines back into many.

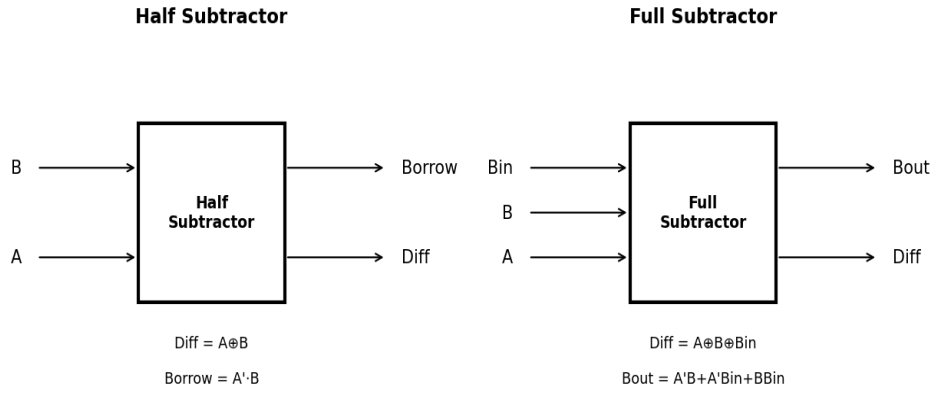
Half Adder & Full Adder



A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

- Half Adder truth table above (2 inputs, no carry-in). **Full Adder:** adds 3 bits (A, B, Cin) — used to build multi-bit adders by cascading (ripple-carry adder).

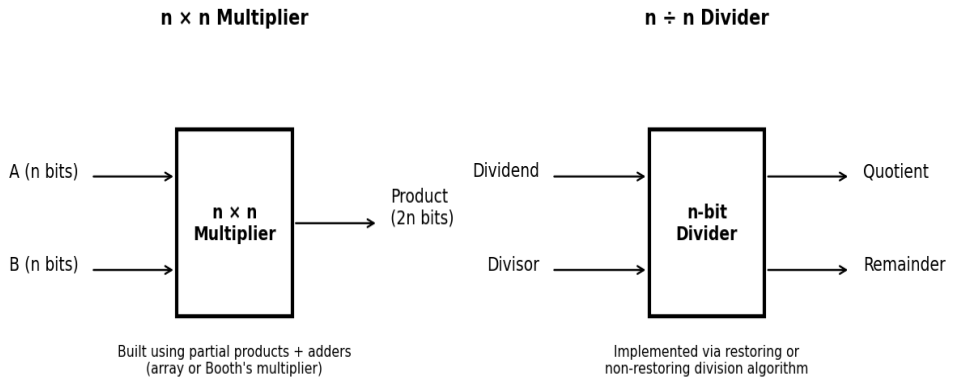
Half Subtractor & Full Subtractor



A	B	Diff	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

- Borrow = 1 only when `A=0, B=1` (i.e. subtracting a larger bit from a smaller one needs a borrow).
- **Binary subtraction in practice:** commonly done by adding the **2's complement** of the subtrahend, avoiding the need for a separate subtractor circuit.

Multiplier & Divider Circuits



- **n×n Multiplier:** commonly implemented using **partial products + adders** — architectures include `array multiplier` (simple, regular structure) and `Booth's multiplier` (efficient for signed numbers, uses shift-and-add with sign detection).
- **n÷n Divider:** implemented using **restoring** or **non-restoring division** algorithms — can be sequential (multi-cycle) or combinational.

> **Easy Explanation**
>
> *Adders/Subtractors are the basis for building ALUs; multipliers extend adders via partial products; dividers extend subtractors via repeated subtraction/shift-and-subtract. All of these arithmetic circuits are still purely COMBINATIONAL (no memory) at the gate level.*

Probable MCQs: Combinational & Arithmetic Circuits

1. A 4:1 multiplexer requires how many select lines?

a) 1
b) 2
c) 3
d) 4
✓Answer: b) 2
Why: $2^n = 4 \rightarrow n = 2$ select lines.

2. The Borrow output of a half subtractor (A−B) is 1 when:

a) A=1, B=0
b) A=0, B=1
c) A=B=1
d) A=B=0
✓Answer: b) A=0, B=1
Why: A borrow is needed only when subtracting a larger bit (B=1) from a smaller one (A=0).

3. The SUM output of a half adder is given by:

a) A·B
b) A+B
c) A⊕B
d) A'B'
✓Answer: c) A⊕B
Why: Half adder Sum = A XOR B (A⊕B).

4. A full adder differs from a half adder because it has an additional:

a) Sum output

b) Carry-in input

c) Carry-out output

d) Select line

✓Answer: b) Carry-in input

Why: Full adder accepts a Carry-in (Cin) from a previous stage, unlike the half adder.

5. Booth's algorithm is primarily associated with which arithmetic circuit?

a) Adder

b) Multiplier (signed numbers)

c) Decoder

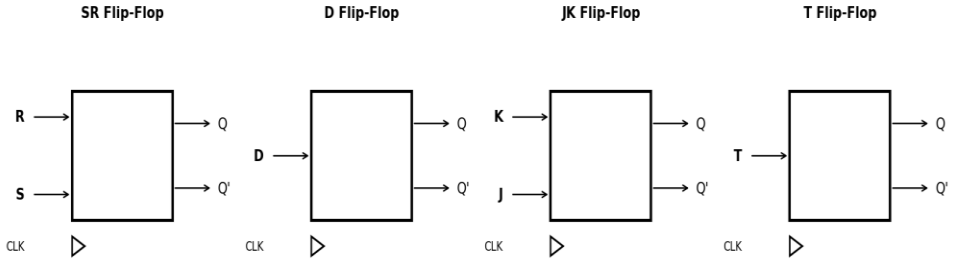
d) Divider only

✓Answer: b) Multiplier (signed numbers)

Why: Booth's algorithm efficiently performs multiplication of signed (2's complement) numbers using shift-and-add.

2.3 Sequential Logic Circuits

Flip-Flop Symbols



SR Flip-Flop — Characteristic Table

S	R	Q (next)	Q' (next)	Operation
0	0	Q (no change)	Q' (no change)	Hold
0	1	0	1	Reset
1	0	1	0	Set
1	1	Undefined	Undefined	Invalid

- S=Set, R=Reset. **S=R=1 is an invalid/forbidden state** in the basic SR flip-flop.

D Flip-Flop — Characteristic Table

D	Q (present)	Q (next)	Q' (next)	Operation
0	Q	0	1	Reset
1	Q	1	0	Set

- Output Q simply **follows D** on the active clock edge — eliminates the invalid state of the SR flip-flop. Widely used in registers.

JK Flip-Flop — Characteristic Table

J	K	Q (present)	Q (next)	Q' (next)	Operation
0	0	Q	Q	Q'	Hold
0	1	Q	0	1	Reset
1	0	Q	1	0	Set
1	1	Q	Q'	Q	Toggle

- J=Set, K=Reset. **J=K=1 makes the output TOGGLE** (complement) — this **removes the invalid state** of the SR flip-flop, making JK more versatile.

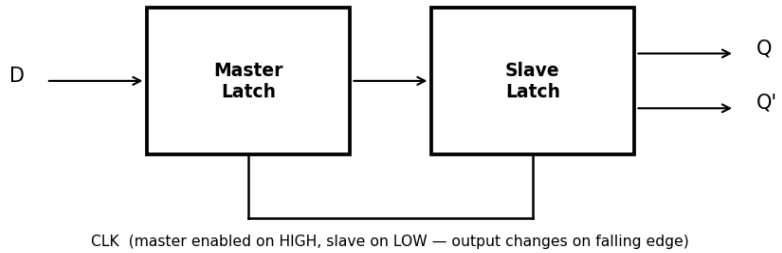
T Flip-Flop — Characteristic Table

T	Q (present)	Q (next)	Q' (next)	Operation
0	Q	Q	Q'	Hold
1	Q	Q'	Q	Toggle

- T = Toggle input. Can be built from a JK flip-flop by tying J=K=T. Very useful in **counters and frequency dividers** (each pulse toggles the output, halving frequency).

Master-Slave Flip-Flop

Master-Slave Flip-Flop Structure

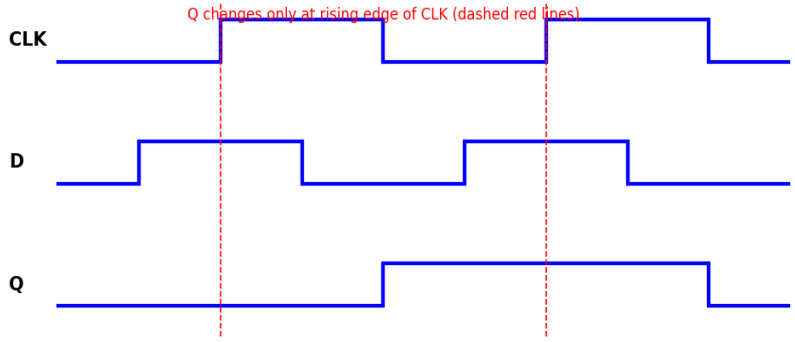


- Uses **two latches in series** (master + slave) — output changes only on the **falling edge** of the clock, which **reduces the race-around problem** and improves noise immunity/stability.

Flip-Flop Quick Comparison

Flip-Flop	Inputs	Characteristic	Common Use
SR	S, R	Set/Reset, invalid at (1,1)	Basic storage, control circuits
D	D	Follows D, no invalid state	Registers, memory
JK	J, K	Toggle at (1,1), no invalid state	Counters, sequential logic
T	T	Toggle when T=1	Counters, frequency dividers
Master-Slave	(varies)	Edge-triggered (falling edge)	High-speed synchronous systems

Positive Edge-Triggered D Flip-Flop — Timing Diagram

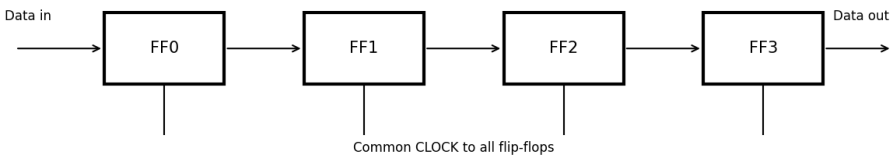


Easy Explanation

Latch = level-triggered (responds while enable is HIGH); Flip-flop = edge-triggered (responds only at the clock transition). Quick recall: SR has an invalid state; D avoids it entirely; JK fixes it by toggling; T is just a simplified JK with J=K=T; Master-Slave improves timing by using two latches and triggering on the falling edge.

Registers & Shift Registers

4-bit Serial-In Serial-Out (SISO) Shift Register



- Register = group of flip-flops storing multiple bits. **Shift register** types by data movement: **SISO, SIPO, PISO, PIPO**.
- Applications: serial-to-parallel conversion, temporary data storage, delay lines.

Counters

- **Asynchronous (Ripple) counter:** FFs triggered by output of previous FF (not common clock) — **simple but slower** due to propagation delay accumulation.
- **Synchronous counter:** all FFs triggered by the **same common clock** simultaneously — **faster, no ripple delay**.

Mod-N counter needs minimum n flip-flops where $2^n \geq N$

Probable MCQs: Sequential Logic

1. In an SR flip-flop, the invalid/forbidden state occurs when:

a) S=0, R=0
b) S=1, R=0
c) S=0, R=1
d) S=1, R=1
✓Answer: d) S=1, R=1
Why: S=R=1 gives an undefined/ambiguous output in a basic SR flip-flop.

2. Which flip-flop's characteristic table has NO invalid/undefined state at all?

a) Basic SR flip-flop
b) JK flip-flop
c) Both have invalid states
d) Neither is a valid flip-flop
✓Answer: b) JK flip-flop
Why: JK flip-flop resolves SR's (1,1) ambiguity by making it a toggle operation instead.

3. A T flip-flop can be constructed from a JK flip-flop by connecting:

a) J=1, K=0 always
b) J and K together as T (J=K=T)
c) T to the clock input
d) J=0, K=1 always
✓Answer: b) J and K together as T (J=K=T)
Why: Tying J and K together to a single input T creates T flip-flop behavior (hold/toggle).

4. A Master-Slave flip-flop changes its output on the:

a) Rising edge only
b) Falling edge of the clock
c) Level (not edge) of the clock
d) Never changes automatically
✓Answer: b) Falling edge of the clock
Why: Master-slave configuration is designed so output updates occur on the falling clock edge, reducing race-around.

5. In an asynchronous (ripple) counter, flip-flops are clocked by:

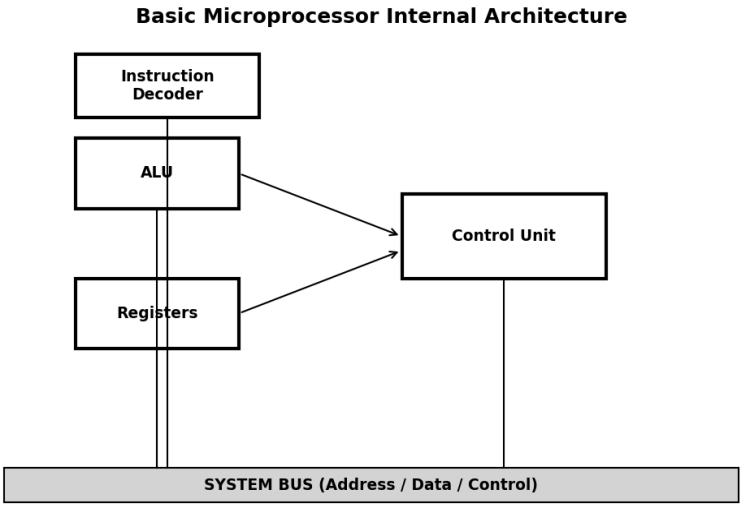
a) A common external clock only
b) The output of the preceding flip-flop
c) Random pulses
d) The power supply

✓Answer: b) The output of the preceding flip-flop

Why: Ripple counters cascade FFs so each one is triggered by the previous FF's output, causing propagation delay.

2.4 Microprocessor

Internal Architecture



- **ALU (Arithmetic Logic Unit):** performs arithmetic (add, subtract) & logical (AND, OR, compare) operations.
- **Control Unit (CU):** generates timing & control signals; directs data flow between components — fetches, decodes, executes instructions.
- **Registers:** small, very fast internal storage — e.g. Accumulator, Program Counter (PC), Instruction Register (IR), Stack Pointer (SP), General-purpose registers.
- **Instruction Decoder:** interprets the opcode fetched and tells the CU which operation to perform.

Basic instruction cycle: FETCH → DECODE → EXECUTE

Easy Explanation

PC (Program Counter) always holds the address of the NEXT instruction to fetch, not the one currently executing.

Assembly Language Programming

- **Low-level language** using mnemonics (e.g. MOV, ADD, JMP) that map almost 1:1 to machine code — needs an **assembler** to convert to machine code.
- Instruction format typically: **Opcode + Operand(s)**. Common addressing modes: immediate, direct, indirect, register, indexed.

Probable MCQs: Microprocessor Architecture

1. Which unit of a microprocessor performs arithmetic and logical operations?

a) Control Unit
b) ALU
c) Program Counter
d) Instruction Decoder
✓Answer: b) ALU
Why: The Arithmetic Logic Unit (ALU) handles all arithmetic and logic computations.

2. The Program Counter (PC) always holds the address of:

a) The last executed instruction

b) The next instruction to be fetched

c) The current data value

d) The stack top

✓Answer: b) The next instruction to be fetched

Why: PC points to the next instruction's address, incremented after each fetch.

3. The correct sequence of a basic instruction cycle is:

a) Decode → Fetch → Execute

b) Fetch → Execute → Decode

c) Fetch → Decode → Execute

d) Execute → Fetch → Decode

✓Answer: c) Fetch → Decode → Execute

Why: Standard instruction cycle order: Fetch, then Decode, then Execute.

4. A program written in assembly language is converted to machine code by:

a) Compiler

b) Assembler

c) Interpreter

d) Linker only

✓Answer: b) Assembler

Why: An assembler translates assembly mnemonics into binary machine code.

5. In an instruction format, the part that specifies the operation to be performed is called:

a) Operand

b) Address field

c) Opcode

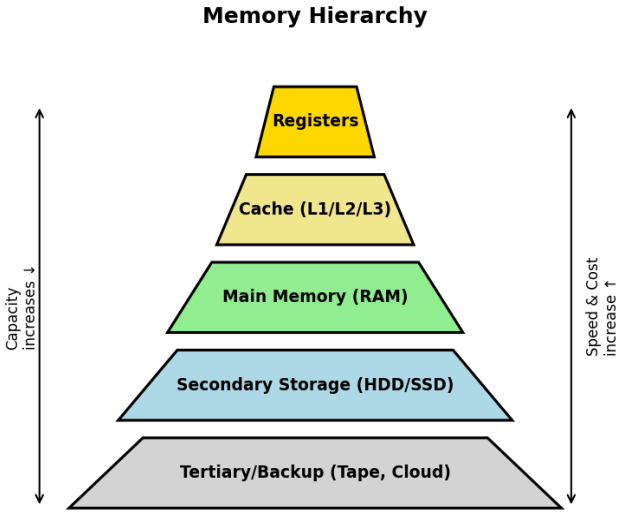
d) Register field

✓Answer: c) Opcode

Why: The Opcode (operation code) specifies WHAT operation to perform; the operand specifies on WHAT data.

2.5 Microprocessor System

Memory Hierarchy



- As we go **up** the hierarchy: **speed ↑, cost/bit ↑, capacity ↓**. As we go **down**: **speed ↓, cost/bit ↓, capacity ↑**.
- **Order (fastest→slowest):** Registers → Cache → Main Memory (RAM) → Secondary Storage (HDD/SSD) → Tertiary/Backup.

I/O and Memory Interfacing

- **Memory-mapped I/O:** I/O devices share the same address space as memory — uses normal memory read/write instructions.
- **I/O-mapped (isolated) I/O:** I/O devices have a **separate address space** — needs special IN/OUT instructions.
- **PPI (Programmable Peripheral Interface, e.g. 8255):** provides parallel I/O ports (Port A, B, C) configurable as input/output for interfacing peripherals.

Serial Interface

- **Synchronous transmission:** data sent in blocks with a shared **clock signal** — higher speed, no start/stop bits, needs precise timing.
- **Asynchronous transmission:** each data character/byte preceded by a **start bit** and followed by **stop bit(s)** — no shared clock, simpler, used in UART/RS-232.
- Common serial standards: **RS-232, USB, SPI, I^2C**.

Direct Memory Access (DMA)

- DMA allows peripherals to transfer data **directly to/from memory without CPU intervention** — greatly speeds up bulk data transfer (e.g. disk, video).
- Controlled by a **DMA Controller (DMAC)** which requests bus control from the CPU (bus request/bus grant handshake).

Easy Explanation

DMA's biggest exam point: it frees the CPU from babysitting data transfer — CPU only sets up the transfer and does other work while DMAC moves the data.

⍰ Probable MCQs: Microprocessor System

1. In the memory hierarchy, as we move from registers towards secondary storage, capacity:

a) Decreases

b) Increases

c) Stays the same

d) Becomes zero

✓Answer: b) Increases

Why: Capacity increases but speed decreases as we move down the hierarchy (registers → ... → secondary storage).

2. In memory-mapped I/O, I/O devices are accessed using:

a) Special IN/OUT instructions only

b) The same instructions used for memory access

c) Interrupts only

d) DMA only

✓Answer: b) The same instructions used for memory access

Why: Memory-mapped I/O shares the address space with memory, so normal memory instructions work.

3. Asynchronous serial transmission uses which bits to frame each character?

a) Parity bits only

b) Start and stop bits

c) Clock bits

d) Sync bits

✓Answer: b) Start and stop bits

Why: Asynchronous transmission frames each byte with a start bit and one or more stop bits.

4. The main advantage of DMA is that it:

a) Increases CPU workload

b) Allows data transfer without continuous CPU involvement

c) Replaces the need for memory

d) Slows down I/O operations

✓Answer: b) Allows data transfer without continuous CPU involvement

Why: DMA offloads bulk data transfer from the CPU, letting the CPU do other tasks simultaneously.

5. A device like the 8255 PPI is primarily used for:

a) Serial communication only

b) Parallel I/O interfacing

c) Memory expansion

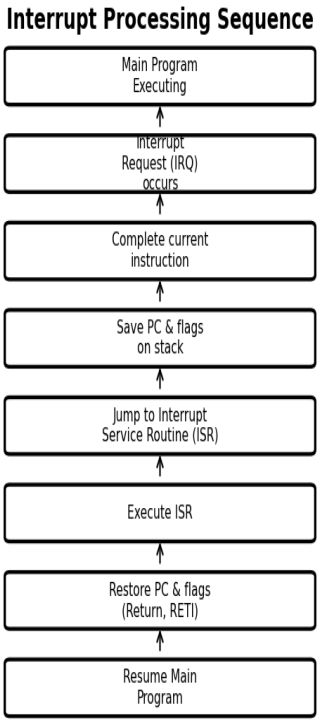
d) Clock generation

✓Answer: b) Parallel I/O interfacing

Why: The 8255 Programmable Peripheral Interface provides configurable parallel I/O ports.

2.6 Interrupt Operations

Interrupts & ISR



- **Interrupt:** a signal that temporarily suspends the main program to service an urgent event/request.
- **ISR (Interrupt Service Routine):** special subroutine that handles the interrupt — CPU jumps here when interrupt is accepted.
- On interrupt: CPU finishes current instruction → **saves PC & flags on stack** → jumps to ISR → executes ISR → **restores PC/flags (RETI)** → resumes main program.

Types of Interrupts

- **Hardware interrupt:** triggered by external hardware/peripheral signal.
- **Software interrupt:** triggered by an instruction within the program (e.g. INT instruction).
- **Maskable interrupt:** can be **disabled/ignored** by the programmer (via interrupt-enable flag).
- **Non-maskable interrupt (NMI):** cannot be disabled — reserved for critical events (e.g. power failure).
- **Vectored interrupt:** automatically jumps to a fixed/pre-defined ISR address.
- **Priority:** when multiple interrupts occur together, the one with **highest priority** is serviced first (handled via priority encoder/interrupt controller).

⍰ Easy Explanation

Polling vs Interrupt: Polling = CPU repeatedly CHECKS each device in turn (wastes CPU time); Interrupt = device ALERTS the CPU only when it needs attention (efficient). This comparison is a favourite MCQ.

⍰ Probable MCQs: Interrupts

1. When an interrupt occurs, the CPU first saves the:

a) Data registers only

b) Program Counter and flags

c) ALU output

d) Stack pointer value only

✓Answer: b) Program Counter and flags

Why: The PC and status flags are pushed onto the stack so the CPU can resume correctly after the ISR.

2. An interrupt that cannot be disabled by software is called:

a) Maskable interrupt

b) Non-maskable interrupt (NMI)

c) Software interrupt

d) Vectored interrupt

✓Answer: b) Non-maskable interrupt (NMI)

Why: NMI is reserved for critical events and always gets serviced, regardless of the interrupt-enable flag.

3. Compared to interrupt-driven I/O, polling is generally:

a) More efficient

b) Less efficient (wastes CPU time)

c) Exactly the same

d) Not possible in microprocessors

✓Answer: b) Less efficient (wastes CPU time)

Why: Polling makes the CPU repeatedly check device status, wasting cycles compared to interrupts.

4. The special instruction used to return from an Interrupt Service Routine is:

a) RET

b) RETI

c) JMP

d) CALL

✓Answer: b) RETI

Why: RETI (Return from Interrupt) restores PC and flags, distinct from a normal subroutine RET.

5. When multiple interrupts occur simultaneously, the CPU services them based on:

a) Alphabetical order

b) Priority

c) Random selection

d) Time of day

✓Answer: b) Priority

Why: Interrupt priority (often via a priority encoder/controller) decides which interrupt is serviced first.