4.2 COMPUTER ORGANISATION & ARCHITECTURE

L	P
4	-

RATIONALE

The subject plays very important role at this level to give exposure to the students about detailed organization of currently available personal computers in order to understand their functioning. It will further help the students in understanding the architecture of computers. The students will also get familiar with multi-processor systems.

COURSE OUTCOMES

After undergoing the subject, the students will be able to:

CO1: Use CPU, register and stack.
CO2: Describe micro programmed and hardwired control.
CO3: Compare RISC and CISC architecture.
CO4: Study memory hierarchy and memory types.
CO5: Explain and perform the functions of BIOS.
CO6: Demonstrate multi-processor systems.

DETAILED CONTENTS

UNIT I

CPU Organisation

General register organisation, stack organisation, instruction formats (three address, two address, one address, zero address and RISC instruction). Addressing modes: Immediate, register, direct, in direct, relative, indexed.

UNIT II

Memory Organisation

Memory Hierarchy, RAM and ROM chips, Memory address map, Memory connections to CPU. Auxiliary memory: Magnetic disks and magnetic tapes. Associative memory, Cache memory, Virtual memory, Memory management hardware, Read and Write operation

UNIT III

I/O Organisation

Basis Input output system (BIOS) - Function of BIOS, Testing and initialization, Configuring the system, Modes of Data Transfer, Programmed I/O: Synchronous, asynchronous and interrupt initiated. DMA data transfer

UNIT IV

Architecture of Multi-processor systems

Forms of parallel processing, Parallel processing and pipelines, basic characteristics of multiprocessor, General purpose multiprocessors, Interconnection networks: time shared common bus, multi-port memory, cross bar switch, multi stage switching networks and hyper cube structures.

UNIT V

I/O Interface

Define I/O interface, Input-Output Interface, Explain methods of Asynchronous Data transfer. Synchronous Data Transfer, Strobe Control, Handshaking, Describe Asynchronous Serial Transfer.

RECOMMENDED BOOKS

1. Computer Architecture and Organisation by Moris Mano.
2. Computer Architecture by J. P. Hayes.
3. Structured Computer Organisation By Tanenbaum Andrew S, PHI.

SUGGESTED WEBSITES

1. http://swayam.gov.in
2. https://ekumbh.aicte-india.org/

INSTRUCTIONAL STRATEGY

This is theoretical subject for basic fundamental knowledge and contains five units of equal weight age.

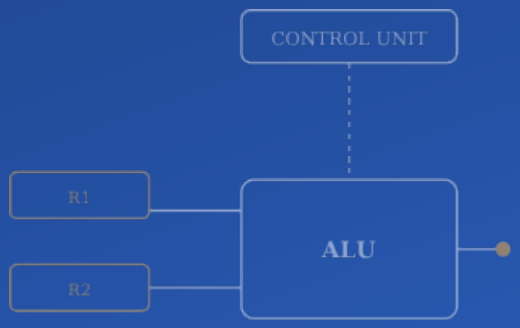
COMPUTER ORGANISATION & ARCHITECTURE

UNIT I

CPU Organisation

General Register Organisation · Stack Organisation · Instruction Formats · Addressing Modes

- Register Organisation
- Stack (Register & Memory)
- 3 / 2 / 1 / 0-Address Formats
- RISC vs CISC
- Addressing Modes



STUDY NOTES | PREPARED FOR EXAM REVISION | UNIT I

UNIT I · OVERVIEW

Table of Contents

How to use these notes: Each topic starts with the core idea, followed by a labelled diagram, working examples, and a boxed summary of advantages/disadvantages — ideal for last-minute revision before exams.

TOPIC 1

General Register Organisation

Early computers used a single **accumulator (AC)** register to hold operands and results, which meant every operation needed a trip to memory. Modern CPUs instead use a set of **general-purpose registers** (R1, R2, ... Rn) connected directly to the ALU, dramatically cutting down memory traffic.

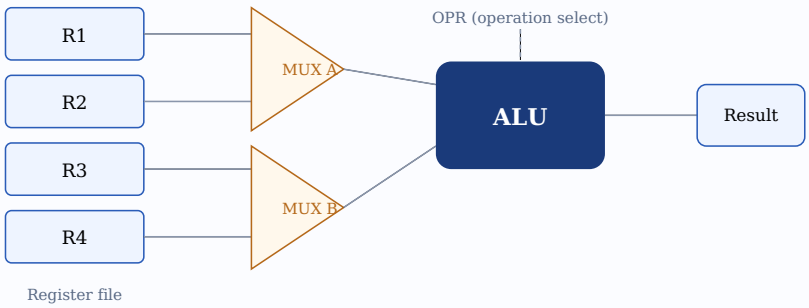


Fig 1.1 — Registers feed two multiplexers (SELA, SELB) that choose ALU inputs; the ALU output (SELD) is written back into a destination register.

Key Components

- A bank of registers (R1–Rn) connected to the ALU inputs and output
- Multiplexers **SELA** and **SELB** to pick the two source registers for the ALU
- **OPR** — operation code that tells the ALU what to compute
- **SELD** — decoder that selects which register receives the ALU result

WHY MULTIPLE REGISTERS?

- Far fewer main-memory accesses (registers are much faster than RAM)
- Speeds up arithmetic / logic operations
- Lets compilers keep frequently-used variables in registers (register allocation)

TOPIC 2

Stack Organisation

A **stack** is a Last-In-First-Out (LIFO) storage structure. Only two operations are allowed on it: **PUSH** (insert at top) and **POP** (remove from top). A pointer called the **Stack Pointer (SP)** always tracks the current top element.

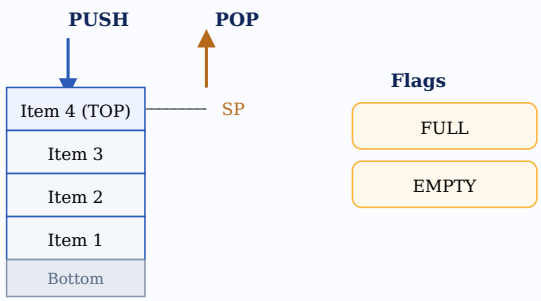


Fig 2.1 — PUSH adds an item at the top and advances SP; POP removes the top item and moves SP back. FULL/EMPTY flags guard against overflow/underflow.

a) Register Stack

- Built from a finite set of registers
- Stack Pointer (SP) points to top register
- FULL and EMPTY flags prevent overflow/underflow
- Fast, but limited in size

b) Memory Stack

- Implemented in a reserved region of RAM
- SP holds the address of the top element
- PUSH: decrement SP, then store data
- POP: read data, then increment SP
- Grows dynamically — limited only by memory size

WHY STACKS MATTER

- Storing return addresses during **subroutine calls / returns**
- Evaluating arithmetic expressions in **postfix (Reverse Polish) notation**
- Forms the basis of the **zero-address instruction format**

TOPIC 3

Instruction Formats

The format depends on how many operand addresses appear explicitly in the instruction. All examples below evaluate: **X = (A + B) × (C + D)**

a) Three-Address Instructions

```
ADD R1, A, B   ; R1 ← A + B
ADD R2, C, D   ; R2 ← C + D
MUL X , R1, R2  ; X  ← R1 × R2
```

Format: OPERATION DEST, SRC1, SRC2 — short program, but longer instruction word.

b) Two-Address Instructions

```
MOV R1, A
ADD R1, B      ; R1 ← R1 + B
MOV R2, C
ADD R2, D      ; R2 ← R2 + D
MUL R1, R2     ; R1 ← R1 × R2
MOV X , R1
```

Result overwrites one of the operands. Common in conventional (x86-style) machines.

c) One-Address Instructions

```
LOAD A     ; AC ← A
ADD  B     ; AC ← AC + B
STORE T    ; T  ← AC
LOAD C     ; AC ← C
ADD  D     ; AC ← AC + D
MUL  T     ; AC ← AC × T
STORE X    ; X  ← AC
```

Uses an implied **Accumulator (AC)** as one operand for every operation.

d) Zero-Address Instructions

Uses a stack; operands come implicitly from the stack top. Expression is first converted to **postfix**: `A B + C D + ×`

```
PUSH A
PUSH B
ADD      ; pop A,B → push (A+B)
PUSH C
PUSH D
ADD      ; pop C,D → push (C+D)
MUL      ; pop top two → push product
POP  X
```

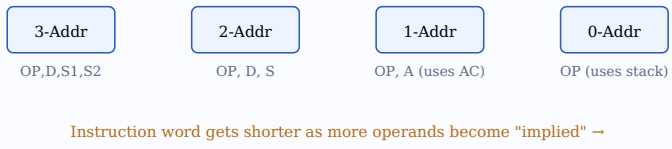


Fig 3.1 — Trade-off between number of explicit addresses and instruction length.

TOPIC 3 (CONTINUED)

e) RISC Instruction Format

RISC (Reduced Instruction Set Computer) uses simple, fixed-length instructions designed for efficient pipelining.

Key Characteristics

- Fixed instruction size → easy to decode and pipeline
- **Load/Store architecture**: only LOAD and STORE touch memory; all other ops work on registers
- Large number of general-purpose registers
- Few, simple addressing modes
- Ideally one instruction executes per clock cycle
- Hardwired control (not microprogrammed) for speed

```
LOAD  R1, A
LOAD  R2, B
ADD   R3, R1, R2   ; register-register operation
STORE R3, X
```

RISC vs CISC

Feature	RISC	CISC
Instruction length	Fixed	Variable
Addressing modes	Few, simple	Many, complex
Memory access	Only via LOAD/STORE	Instructions can operate directly on memory
Control unit	Hardwired	Microprogrammed
Registers	Many	Fewer
Example	ARM, MIPS	x86

> **Exam tip:** If asked to compare, always mention the *load/store principle* for RISC — it's the single most examined distinguishing feature.

TOPIC 4

Addressing Modes

Addressing modes define **how the operand's location is specified** in an instruction.

Mode	Description	Example	Effective Address (EA)
Immediate	Operand given directly in the instruction	MOV R1, #5	Operand = 5 (no memory access)
Register	Operand is inside a register	MOV R1, R2	EA = R2
Direct	Address of operand given directly	MOV R1, [1000]	EA = 1000
Indirect	Address field points to a location that holds the operand's address	MOV R1, @[1000]	EA = content of location 1000
Relative	EA = Program Counter (PC) + offset	MOV R1, 20(PC)	EA = PC + 20
Indexed	EA = Index Register + offset	MOV R1, 20(R2)	EA = R2 + 20

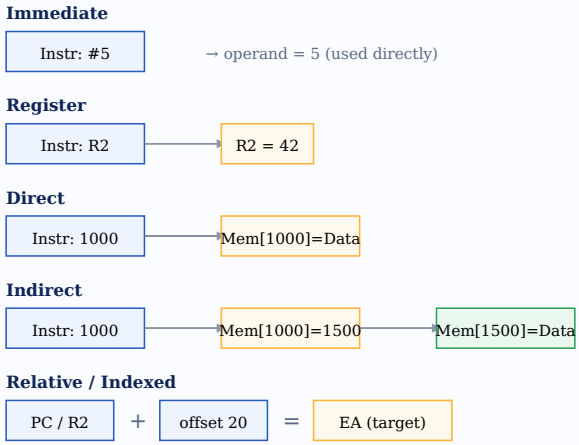


Fig 4.1 — How the effective address (EA) is resolved differently in each mode.

Quick Notes

- **Immediate:** fastest, no memory access; limited by instruction length
- **Register:** fast, no memory access; limited by number of registers
- **Direct:** one memory access; addressing range limited by address-field size
- **Indirect:** two memory accesses (slower), but reaches a much larger address space
- **Relative:** EA relative to PC — enables position-independent code; used in branches
- **Indexed:** ideal for arrays — index register increments while offset (base) stays fixed

TOPIC 5

Quick Revision Summary

One-Glance Recap

Concept	Core Idea
General Register Organisation	Multiple registers + ALU + MUX (SELA/SELB) + decoder (SELD), controlled by OPR
Register Stack	Fixed set of registers, SP + FULL/EMPTY flags
Memory Stack	Stack implemented in RAM; SP tracks top; grows dynamically
3-Address	OP, D, S1, S2 — shortest program, longest instruction
2-Address	OP, D, S — result overwrites one operand
1-Address	OP, A — uses implied Accumulator (AC)
0-Address	OP only — uses stack; needs postfix notation
RISC	Fixed-length, load/store only, hardwired control, many registers
Immediate mode	Operand in instruction itself
Register mode	Operand in a register
Direct mode	Address field = operand address
Indirect mode	Address field → pointer → operand (2 memory accesses)
Relative mode	EA = PC + offset
Indexed mode	EA = Index register + offset

Golden rule for instruction formats: more explicit addresses = fewer instructions per program but a longer instruction word; fewer explicit addresses (as in 0-address/stack machines) = shorter instructions but more of them.

COMMONLY ASKED EXAM QUESTIONS

- Write 3-address, 2-address, 1-address and 0-address code for a given expression
- Differentiate RISC vs CISC
- Explain register stack vs memory stack with diagram
- Compute the effective address for each addressing mode given sample values

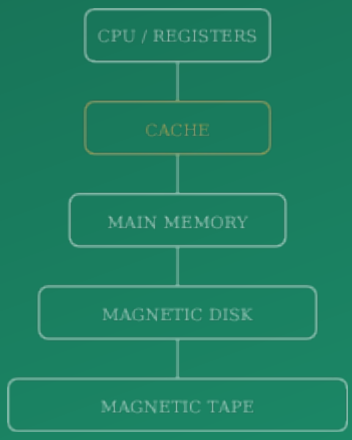
COMPUTER ORGANISATION & ARCHITECTURE

UNIT II

Memory Organisation

Memory Hierarchy · RAM/ROM · Address Maps · Cache & Virtual Memory · Auxiliary Storage

- Memory Hierarchy
- RAM & ROM Chips
- Memory Address Map
- Auxiliary Memory
- Associative Memory
- Cache Memory
- Virtual Memory



UNIT II · OVERVIEW

Table of Contents

How to use these notes: Each topic opens with the core idea, a labelled diagram, and a boxed summary of key points — built for quick last-minute revision before exams.

TOPIC 1

Memory Hierarchy

No single memory technology is fast, large, and cheap all at once. So computer systems arrange memory in a **hierarchy**: the fastest, most expensive, smallest memories sit closest to the CPU; the slowest, cheapest, largest memories sit farthest away.

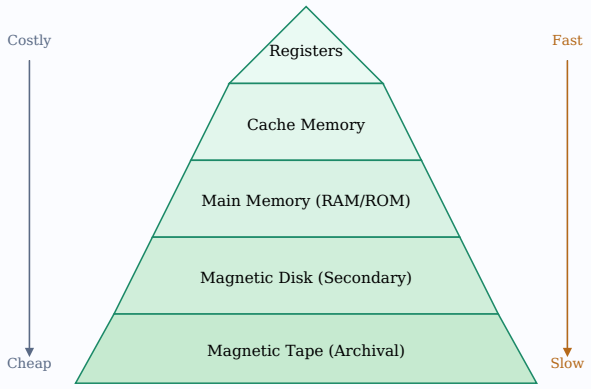


Fig 1.1 — Memory hierarchy pyramid: speed and cost decrease, capacity increases, going down.

Levels (top to bottom)

- **CPU Registers** — fastest, smallest, most expensive per bit
- **Cache Memory** — very fast buffer between CPU and main memory
- **Main Memory (RAM/ROM)** — directly addressable by the CPU
- **Magnetic Disk** — secondary storage, large capacity, non-volatile
- **Magnetic Tape** — slowest, cheapest, used for backup/archival

WHY A HIERARCHY?

- Balances the conflicting goals of speed, capacity, and cost
- Exploits **locality of reference** — programs repeatedly access nearby data/instructions
- Keeps frequently used data in fast memory, rarely used data in slow/cheap memory

TOPIC 2

RAM and ROM Chips

Main memory is built from semiconductor chips of two broad types: **RAM** (Random Access Memory, read/write, volatile) and **ROM** (Read Only Memory, non-volatile).

RAM

- Both read and write operations possible
- **Volatile** — loses content when power is off
- **Static RAM (SRAM):** flip-flop based, fast, no refresh needed, costly — used in cache
- **Dynamic RAM (DRAM):** capacitor based, needs periodic refresh, cheaper, denser — used as main memory

ROM

- Normally read-only; used to hold permanent programs (e.g. bootstrap loader)
- **Non-volatile** — retains data without power
- **PROM:** programmed once by the user
- **EPROM:** erasable with UV light, reprogrammable
- **EEPROM:** erasable electrically, reprogrammable in-circuit

Fig 2.1 — RAM has bidirectional data lines (read/write); ROM has one-directional data lines (read only), both use address lines and a chip-select line.

Chip capacity: A chip with k address lines and n data lines has 2^k words, each of n bits — e.g. a chip with 10 address lines and 8 data lines is a 1K × 8 memory chip.

TOPIC 3

Memory Address Map & Connection to CPU

When several RAM and ROM chips are connected to a CPU, each chip must be assigned a unique range of addresses. A **memory address map** is a table showing which address ranges belong to which chip, and it guides how the **chip-select (CS)** logic is wired using the CPU's address lines.

Example: 512-byte Memory (128B RAM × 2 + 256B ROM)

Component	Address Range (Hex)	Address Bus Lines (A8 A7 A6...A0)
RAM 1 (128B)	0000 – 007F	0 0 x x x x x x x
RAM 2 (128B)	0080 – 00FF	0 1 x x x x x x x
ROM (256B)	0100 – 01FF	1 x x x x x x x x

The high-order address bits are decoded to generate each chip's **chip-select** signal, while the low-order bits directly index a specific word inside the selected chip.

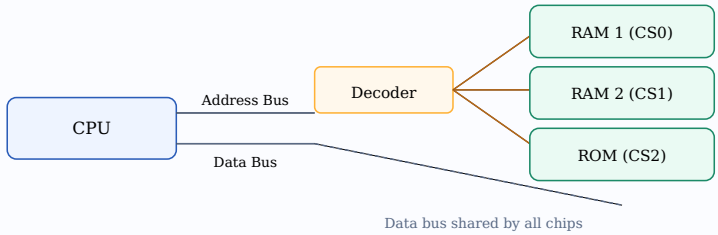


Fig 3.1 — CPU's address bus feeds a decoder that activates one chip-select line at a time; the data bus is shared among all chips.

TOPIC 4

Auxiliary Memory

Auxiliary (secondary) memory provides large, low-cost, non-volatile storage outside the CPU-memory path. The two classic devices are **magnetic disks** and **magnetic tapes**.

a) Magnetic Disks

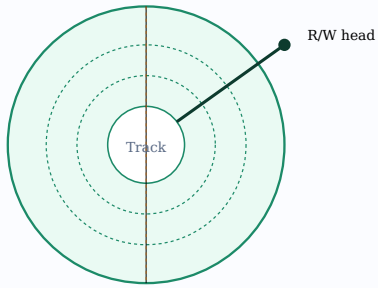


Fig 4.1 — Disk surface organised into concentric tracks; each track divided into sectors; a movable read/write head accesses data directly (random access).

- Data stored as magnetised spots on concentric **tracks**, each split into **sectors**
- **Direct-access** device — access time = seek time + rotational latency + transfer time
- Used for main secondary storage (OS, files, databases)

b) Magnetic Tapes

- **Sequential-access** device — must read through preceding data to reach a location
- Data recorded on a magnetic-coated plastic tape wound on reels
- Very high capacity, low cost per bit; slow access
- Used mainly for **backup and archival** storage

Disk vs Tape: Disks give random/direct access (fast for everyday use); tapes give sequential access only (cheap, ideal for backups, not for frequent access).

TOPIC 5

Associative Memory

Also called **Content Addressable Memory (CAM)**, associative memory is accessed by its **content** rather than by its address. You supply a search argument, and all locations that match are found *simultaneously*, in parallel.

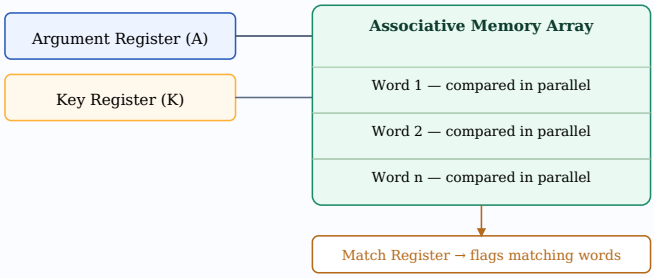


Fig 5.1 — The Key register masks which bits of the Argument register are compared; all words are searched in parallel and matches are flagged.

Key Points

- **Argument register (A):** holds the value to be searched for
- **Key register (K):** selects which bits of A participate in the comparison (masking)
- Every word compares itself with A *simultaneously* using built-in logic per cell — hence "content addressable"
- **Match register:** a bit per word, set to 1 for each matching word

APPLICATIONS

- Cache memory (tag comparison)
- Database search / lookup tables
- Address translation in virtual memory systems

TOPIC 6

Cache Memory

Cache is a small, very fast memory placed between the CPU and main memory. It holds copies of recently used instructions/data so that the CPU can access them quickly, exploiting the **locality of reference** principle.

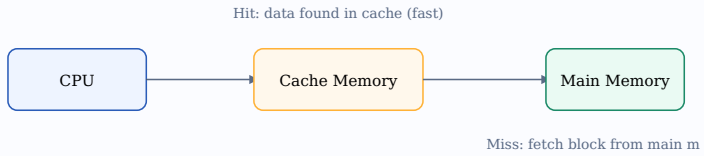


Fig 6.1 — Cache sits between CPU and main memory; a "hit" is served fast, a "miss" pulls a block from main memory into the cache.

Mapping Techniques

Technique	Idea
Direct Mapping	Each main-memory block maps to exactly one specific cache line (address MOD cache size)
Associative Mapping	A block can go into any cache line; requires comparing tags in parallel (uses associative memory)
Set-Associative Mapping	Cache divided into sets; a block maps to any line within one specific set — a compromise between the two above

Key Terms

- **Hit ratio:** fraction of memory accesses found in cache
- **Locality of reference:** temporal (reused soon) and spatial (nearby addresses used soon)
- **Replacement policy:** decides which block to evict on a miss (e.g. LRU — Least Recently Used)
- **Write policy:** Write-through (write to cache and memory together) vs Write-back (write to cache only, update memory later)

TOPIC 7

Virtual Memory & Memory Management Hardware

Virtual memory gives programs the illusion of a large, contiguous address space even though physical main memory may be smaller. Programs use **logical (virtual) addresses**, which are translated to **physical addresses** at run time by memory management hardware.

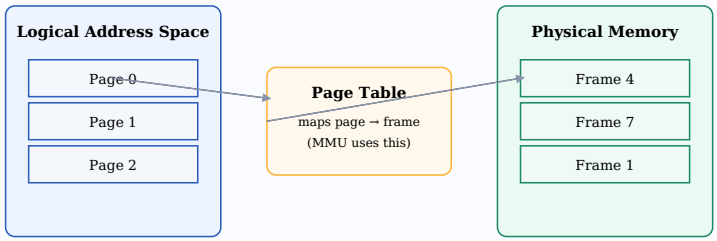


Fig 7.1 — Paging: logical address space is divided into pages; the page table (managed by the MMU) maps each page to a physical frame.

Techniques

- **Paging:** logical memory divided into fixed-size *pages*; physical memory divided into equal-size *frames*; a page table maps pages to frames
- **Segmentation:** logical memory divided into variable-size *segments* (e.g. code, data, stack), each with its own base and limit
- **Memory Management Unit (MMU):** hardware that performs the logical-to-physical address translation on every memory reference

BENEFITS OF VIRTUAL MEMORY

- Programs can be larger than available physical memory
- Enables multiprogramming — several programs share memory safely (protection via page tables)
- Simplifies memory allocation for the programmer

TOPIC 8

Read and Write Operations

Every memory access is either a **Read** (fetch data without altering it) or a **Write** (store new data, destroying the old value at that location).

```
READ cycle:
  1. CPU places the address on the Address Bus
  2. CPU asserts the Read control line
  3. Memory places the addressed word on the Data Bus
  4. CPU reads the data from the Data Bus

WRITE cycle:
  1. CPU places the address on the Address Bus
  2. CPU places the data to be stored on the Data Bus
  3. CPU asserts the Write control line
  4. Memory stores the data from the Data Bus into the addressed location
```

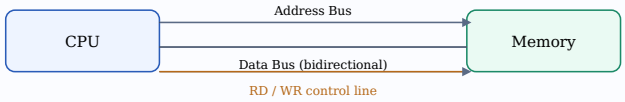


Fig 8.1 — Address bus (CPU → memory), Data bus (bidirectional), and a control line selecting Read or Write.

Quick Revision Summary

Concept	Core Idea
Memory Hierarchy	Registers → Cache → Main Memory → Disk → Tape (speed ↓, capacity ↑)
RAM	Volatile, read/write; SRAM (fast, cache) vs DRAM (dense, main memory)
ROM	Non-volatile, read-only; PROM/EPROM/EEPROM variants
Memory Address Map	Table assigning address ranges to chips; decoder drives chip-select lines
Magnetic Disk	Direct access; tracks + sectors; used for active secondary storage
Magnetic Tape	Sequential access; cheap, high capacity; used for backup/archival
Associative Memory	Content-addressable; parallel search using Argument & Key registers
Cache Memory	Small fast buffer; direct / associative / set-associative mapping
Virtual Memory	Logical → physical translation via paging/segmentation using the MMU
Read/Write	Read: memory → CPU over data bus; Write: CPU → memory over data bus

Exam tip: Diagram-based questions are common for cache mapping techniques and the paging/MMU address-translation process — practice drawing both from memory.

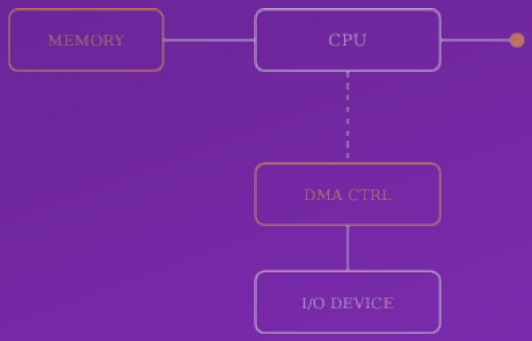
COMPUTER ORGANISATION & ARCHITECTURE

UNIT III

I/O Organisation

BIOS · Modes of Data Transfer · Programmed, Interrupt-Initiated & DMA I/O

- BIOS Functions
- Testing & Initialization (POST)
- System Configuration
- Programmed I/O
- Interrupt-Initiated I/O
- DMA Data Transfer



STUDY NOTES | PREPARED FOR EXAM REVISION | UNIT III

UNIT III · OVERVIEW

Table of Contents

How to use these notes: Each topic opens with the core idea, a labelled diagram, and a boxed summary of key points — built for quick last-minute revision before exams.

TOPIC 1 & 2

Basic Input Output System (BIOS)

The **BIOS** is firmware stored in a ROM/Flash chip on the motherboard. It is the first program that runs when a computer is powered on, acting as a bridge between the hardware and the operating system.

Fig 1.1 — Boot sequence: Power on → BIOS runs from ROM → POST & device initialization → bootstrap loader hands control to the Operating System.

Functions of BIOS

- **Power-On Self-Test (POST):** checks that hardware (CPU, RAM, keyboard, disk drives, etc.) is present and working before boot
- **Initialization:** sets up hardware registers and identifies/activates devices needed for boot
- **Bootstrap loader:** locates the operating system on a storage device and loads it into memory
- **BIOS setup / configuration:** lets the user view and change hardware settings (boot order, date/time, enabling/disabling devices)
- **Provides a hardware abstraction layer:** low-level routines (interrupts) that the OS/programs can call to access hardware, without knowing hardware-specific details

WHY BIOS MATTERS

- Ensures the machine is safe to boot (hardware faults are caught early by POST)
- Provides a standard interface so the OS can work across different hardware
- Stored in non-volatile memory so it survives power loss and runs before any OS exists

TOPIC 3

Testing and Initialization (POST)

The **Power-On Self-Test (POST)** is a diagnostic sequence the BIOS runs immediately after power-on, before the operating system is loaded.

Fig 3.1 — POST checks core hardware components in sequence; failures are reported via beep codes or on-screen messages before boot continues (or halts).

What POST Checks

- **CPU:** verifies the processor responds and basic instructions execute correctly
- **Memory (RAM):** tests readability/writability of installed memory chips
- **Keyboard / input devices:** checks that input devices are connected and responding
- **Disk drives / storage controllers:** verifies storage devices are detected
- **Video card / display:** initializes the display adapter so error messages can be shown

Initialization

After successful testing, BIOS **initializes** hardware: it sets registers on I/O controllers, assigns system resources (IRQs, I/O addresses), and prepares devices to be used by the operating system, before finally invoking the **bootstrap loader**.

> **Exam tip:** POST failures are commonly indicated by a pattern of *beep codes* when the display itself isn't yet functional — a frequently asked point.

TOPIC 4

Configuring the System

BIOS provides a **setup utility** (accessed by pressing a key like Del/F2 during boot) that lets users view and modify hardware configuration settings. These settings are stored in a small battery-backed memory called **CMOS RAM** so they persist even when the computer is off.

Setting	Purpose
Boot order	Decides which device (HDD, USB, network) BIOS tries first to load an OS from
System date/time	Sets the real-time clock used by the OS
Enable/disable devices	Turns onboard components (USB ports, audio, network) on or off
Memory/CPU settings	Configures clock speeds, voltages (overclocking), memory timings
Security	Sets BIOS/supervisor passwords, Secure Boot options
Power management	Configures sleep/wake behaviour

CMOS vs BIOS

- **BIOS:** the firmware program itself, stored in ROM/Flash memory
- **CMOS RAM:** small volatile memory that stores the BIOS's configuration settings, kept alive by a small battery (the "CMOS battery") on the motherboard

TOPIC 5

Modes of Data Transfer — Overview

Data moves between the CPU/memory and I/O devices using one of three broad techniques, each trading off CPU involvement against speed and hardware complexity.

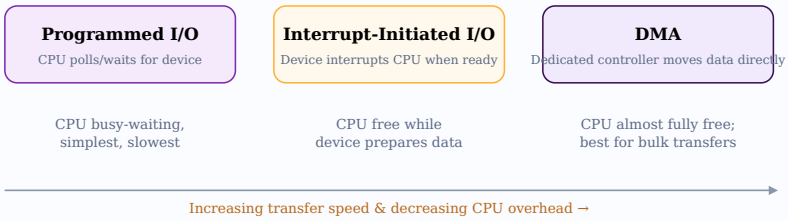


Fig 5.1 — The three main modes of data transfer, ordered by CPU involvement.

Mode	CPU Role	Best Suited For
Programmed I/O	Actively polls device status in a loop; fully occupied	Simple systems, small/infrequent transfers
Interrupt-Initiated I/O	Does other work; device signals via interrupt when ready	Moderate-speed devices (keyboard, mouse)
DMA	Only sets up the transfer; DMA controller handles the rest	High-speed, bulk transfers (disk, network)

TOPIC 6

Programmed I/O

In **Programmed I/O**, all data transfer is controlled directly by a CPU program: the CPU repeatedly checks (polls) the device's status register and moves data one word at a time through the accumulator/registers.

```
LOOP:  READ  STATUS_REG      ; check device status
       TEST  READY_BIT
       JUMP_IF_NOT_READY LOOP  ; busy-wait / poll
       READ  DATA_REG        ; transfer one word
       STORE DATA, MEMORY
```

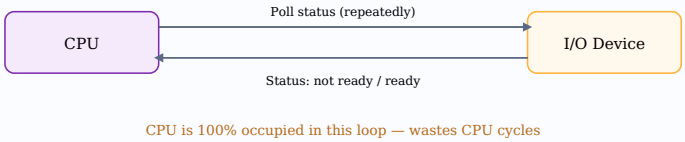


Fig 6.1 — CPU continuously polls the device status register; this busy-waiting wastes CPU time but requires no extra hardware.

Synchronous vs Asynchronous Data Transfer

Type	Description
Synchronous	CPU and device share a common clock; transfer happens at fixed, known time intervals — simple but requires devices of similar speed
Asynchronous	No common clock; uses a **handshaking** protocol (Ready/Busy or Request/Acknowledge signals) so devices of different speeds can communicate reliably

Asynchronous Handshaking (example: Source-Initiated)

1. Source sets data on the bus and activates a **"Data Valid"** signal
2. Destination accepts the data and activates **"Data Accepted"**
3. Source disables "Data Valid" upon seeing "Data Accepted"
4. Destination disables "Data Accepted", ready for the next transfer

Interrupt-Initiated I/O

Instead of polling, the CPU issues a command to the device and continues with other tasks. When the device finishes preparing data, it sends an **interrupt signal** to the CPU, which then suspends its current work, transfers the data (via an interrupt service routine), and resumes.

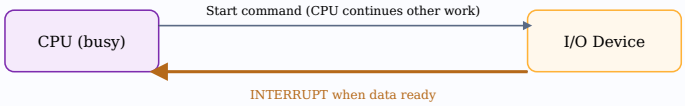


Fig 6.2 — CPU starts the device and continues other work; the device interrupts the CPU only when it needs attention, freeing up CPU time compared to polling.

TOPIC 7

DMA Data Transfer

Direct Memory Access (DMA) allows an I/O device to transfer data directly to/from main memory without CPU intervention for each word, using a dedicated **DMA controller**.

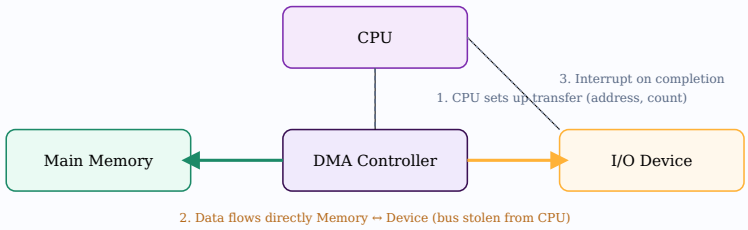


Fig 7.1 — CPU programs the DMA controller with the memory address, device address, and word count, then continues other work; the DMA controller performs the actual transfer and interrupts the CPU only when done.

DMA Controller Registers

- **Address register:** holds the memory address for the next word to transfer
- **Word count register:** holds the number of words remaining to transfer
- **Control register:** specifies transfer mode (read/write) and status

DMA Transfer Modes

Mode	Description
Burst / Block transfer	DMA controller takes control of the bus and transfers an entire block of data before releasing the bus back to the CPU
Cycle stealing	DMA controller transfers one word at a time, "stealing" one bus cycle between CPU cycles, so the CPU is only slightly slowed rather than fully halted

WHY DMA IS FASTER

- CPU is involved only at the start (setup) and end (interrupt) of the transfer
- Data moves directly between device and memory, bypassing CPU registers entirely
- Ideal for high-volume transfers: disk I/O, video, network

Exam tip: Be ready to compare all three modes (Programmed / Interrupt / DMA) in a single table — a very common exam question.

TOPIC 8

Quick Revision Summary

Concept	Core Idea
BIOS	Firmware in ROM; runs first at power-on; bridges hardware and OS
Functions of BIOS	POST, initialization, bootstrap loading, setup/configuration, hardware abstraction
POST	Tests CPU, RAM, input devices, storage, video before boot; reports failures via beep codes
System Configuration	BIOS setup utility; settings stored in battery-backed CMOS RAM
Programmed I/O	CPU polls device status in a loop; simple but wastes CPU time
Synchronous transfer	Common clock between CPU and device; fixed timing
Asynchronous transfer	Handshaking signals (Data Valid / Data Accepted) instead of a common clock
Interrupt-Initiated I/O	CPU does other work; device interrupts CPU when ready
DMA	Dedicated controller transfers data directly between memory and device; CPU involved only at start/end
DMA modes	Burst/block transfer vs cycle stealing

Golden rule: As you move from Programmed → Interrupt-Initiated → DMA, CPU involvement decreases and effective transfer speed for bulk data increases.

COMMONLY ASKED EXAM QUESTIONS

- Explain the functions of BIOS with a boot sequence diagram
- Differentiate synchronous, asynchronous, and interrupt-initiated data transfer
- Explain DMA data transfer with a block diagram and its transfer modes
- Compare Programmed I/O, Interrupt I/O, and DMA in a table

STUDY NOTES

CO

Computer Organization

UNIT-4

Architecture of Multiprocessor Systems — forms of parallel processing, multiprocessor characteristics, general-purpose multiprocessors, and the interconnection networks that tie them together.

DISCIPLINE	Computer Engineering
SEMESTER	4th
SUBJECT	Computer Organization
TOPICS COVERED	Parallel Processing · Multiprocessors · Interconnection Networks
SUGGESTED DURATION	14 Hours

Table of Contents

1 Forms of Parallel Processing

Parallel processing is a general term for techniques that get more than one operation done at the same time, rather than one instruction finishing before the next begins. It can be applied at many different levels inside a computer system — not only by adding more processors, but also by restructuring how a single processor handles work. These levels are usually grouped into six recognised forms.

1. Multiplicity of Functional Units

Early single-ALU processors performed one arithmetic or logic operation at a time. Modern CPUs instead provide several independent functional units — separate units for integer addition, multiplication, floating-point arithmetic, logical operations, and shifting. Because these units are independent, the control unit can dispatch different operations to different units in the same cycle, so several instructions can be "in progress" simultaneously rather than queued one behind another.

2. Parallelism and Pipelining within the CPU

Arithmetic operations can themselves be broken into pipelined stages (as covered under floating-point and instruction pipelines), and a single functional unit can be internally parallel — for example, a carry-look-ahead adder resolves all bit positions together instead of rippling a carry through the circuit one bit at a time.

3. Overlapped CPU and I/O Operations

I/O operations can proceed concurrently with CPU computation once the transfer has been initiated — using techniques such as interrupt-driven I/O or DMA (as covered in Unit 3). This overlap means the CPU is never left idle purely waiting for a slow peripheral to finish a transfer.

4. Use of a Hierarchical Memory System

Memory is organised into a hierarchy — registers, cache, main memory, and secondary storage — so that the fast processor is not constantly stalled waiting on the slowest storage device. Data likely to be reused is kept closer to the CPU, which keeps functional units supplied with operands and instructions.

5. Balancing of Subsystem Bandwidth

A system is only as fast as its slowest link. Balancing subsystem bandwidth means matching the data rate of memory, I/O channels, and the CPU itself so that no single subsystem becomes a bottleneck that starves the others of work.

6. Multiprogramming and Time-Sharing

At the operating-system level, multiprogramming keeps several programs resident in memory at once and switches the CPU between them so it is never left idle while one program waits on I/O. Time-sharing extends this idea to give many interactive users the impression of simultaneous access to the same machine.

MULTIPLE PROCESSOR SYSTEMS

Beyond restructuring a single CPU, parallelism can also be achieved by connecting multiple processors together — either **loosely coupled** (each with its own memory, communicating over a network) or

> **tightly coupled** (sharing a common memory). This is the form of parallel processing this unit focuses on: multiprocessor architecture.

2 Parallel Processing and Pipelines

Pipelining and multiprocessing are two different routes to the same goal — higher throughput — and it helps to be clear about how they differ.

Aspect	Pipelining	Multiprocessing
Type of parallelism	Temporal — different stages of different instructions overlap in time on shared hardware	Spatial — different processors execute different instructions on different hardware at the same physical moment
Hardware	A single processing unit divided into stages	Multiple independent processing units
Granularity	Fine-grained — operates at the level of individual instructions	Coarse-grained — operates at the level of whole tasks, processes, or large blocks of instructions
Scaling limit	Bounded by the number of pipeline stages and by hazards (data, control, structural)	Bounded by interconnection bandwidth, memory contention, and how well the workload can be divided

The two techniques are complementary rather than competing: a multiprocessor system's individual processors are themselves typically pipelined internally. A pipeline increases the throughput of a single instruction stream; multiprocessing increases throughput by running several instruction streams side by side. High-performance systems use both together — pipelined processors, several of them, interconnected into a single multiprocessor system.

> *Pipelining parallelises the stages of one job. Multiprocessing parallelises the jobs themselves.*

3 Basic Characteristics of Multiprocessors

DEFINITION

A multiprocessor is a computer system with two or more CPUs that are capable of communicating and cooperating with one another, at different levels, to solve a given problem.

Multiprocessors are distinguished from simple collections of independent computers by several shared characteristics:

1. **Multiple processing units.** Two or more CPUs, each with its own control unit and set of functional units, are present in the same system.
2. **Shared resources.** The processors share access to some combination of main memory, I/O devices, and peripheral controllers, rather than operating as fully independent machines.
3. **Interconnection structure.** A communication path — a bus, switch, or network — links the processors to each other and to shared memory. The choice of interconnection structure directly affects how well the system scales (see Section 5).
4. **A single integrated operating system.** Unlike a network of independent computers, a true multiprocessor is normally managed by one operating system (or a small number of tightly cooperating kernels) that schedules processes across all available CPUs and coordinates their access to shared resources.
5. **Common goal.** All processors work toward a common computational goal: either executing parts of the same program concurrently, or running independent jobs whose combined throughput benefits from shared hardware.

Loosely Coupled vs. Tightly Coupled Systems

Multiprocessor systems are broadly classified by how closely their processors share memory:

Tightly Coupled	Loosely Coupled
Processors share a common, global main memory.	Each processor has its own private (local) memory.
Communication between processors happens through shared variables in that common memory — very fast, but requires careful synchronisation to avoid conflicts.	Communication happens by passing messages over an interconnection network — slower per exchange, but avoids memory-access contention.
Typically a small number of processors, physically close together (e.g. a single multi-core chip or board-level SMP system).	Scales to a much larger number of processors, which may be physically distributed (e.g. clusters, distributed systems).
Best suited to fine-grained parallel tasks that need to exchange data frequently.	Best suited to coarse-grained parallel tasks that can run largely independently.

4 General-Purpose Multiprocessors

A general-purpose multiprocessor is designed to run a broad mix of workloads — scientific computation, transaction processing, general time-sharing — rather than being built for one narrow task. General-purpose multiprocessor systems are usually organised in one of three ways:

Master–Slave Configuration

One processor — the master — is designated to run the operating system, handle scheduling decisions, and manage I/O. The remaining processors act as slaves, executing user (application) code only when dispatched by the master. This is simple to implement, since only the master needs to deal with operating-system complexity, but the master itself can become a bottleneck as the number of slave processors grows.

Separate (Independent) Operating Systems

Each processor runs its own complete, independent copy of the operating system. The processors may share some hardware resources (such as I/O devices), but scheduling and memory management are handled locally by each processor's own kernel. This avoids a single bottleneck but complicates coordination when processors do need to cooperate on a shared task.

Symmetric (or "Floating Control") Multiprocessing — SMP

All processors are treated as equals: any processor can execute operating-system code, service interrupts, and run user programs. A single copy of the operating system resides in shared memory, and processors coordinate through shared data structures protected by locks. Because control is not fixed on one processor, the system load balances naturally across all CPUs. This is the organisation used in most modern general-purpose multiprocessor and multi-core systems.

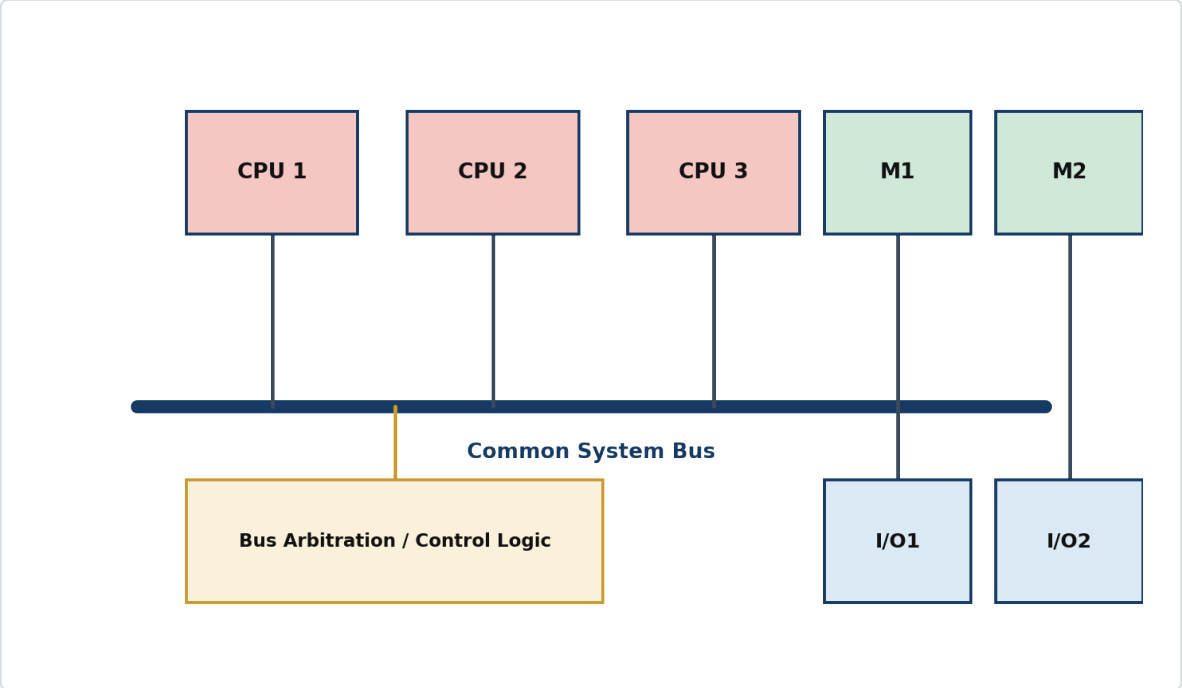
Master–Slave **Separate OS per CPU** **Symmetric (SMP)**

5 Interconnection Networks

Whatever the coupling scheme, every multiprocessor needs a physical structure that connects its processors to memory, to I/O, and to each other. This interconnection network has a direct impact on the system's cost, its maximum size, and how much contention processors experience when they compete for shared resources. Five classic interconnection schemes are described below, in roughly increasing order of hardware complexity and achievable performance.

Time-Shared Common Bus

In this scheme, every processor, memory module, and I/O unit is attached to one shared communication path — the system bus. At any instant, only one device may transmit on the bus, so a bus arbitration (control) unit grants access to whichever requester is selected, based on a priority or fairness scheme, before that transfer can begin.

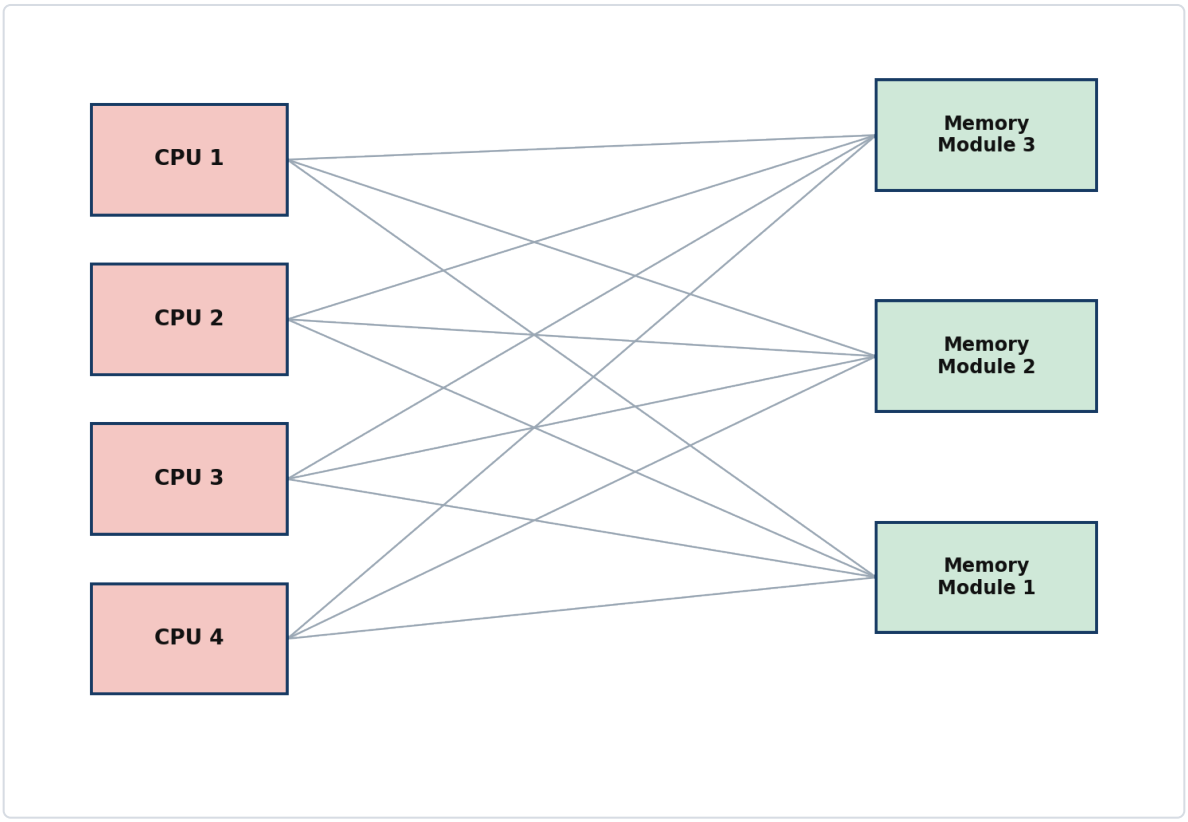


Time-shared common bus: all CPUs, memory modules, and I/O units are attached to a single shared bus, mediated by an arbitration unit.

Advantages	Disadvantages
Simplest and cheapest interconnection scheme to build.	Only one transfer can occur at a time — the bus quickly becomes a bottleneck as more processors are added.
Easy to expand by attaching additional devices to the same bus.	Bandwidth is shared among all attached devices, so per-processor performance degrades as the system grows.

Multiport Memory

Instead of sharing a single bus, each memory module in a multiport memory system is given a separate physical port for every CPU that needs to access it. Each processor therefore has a private, dedicated path to every memory module, and the memory module itself contains the control logic needed to resolve simultaneous requests arriving on different ports.

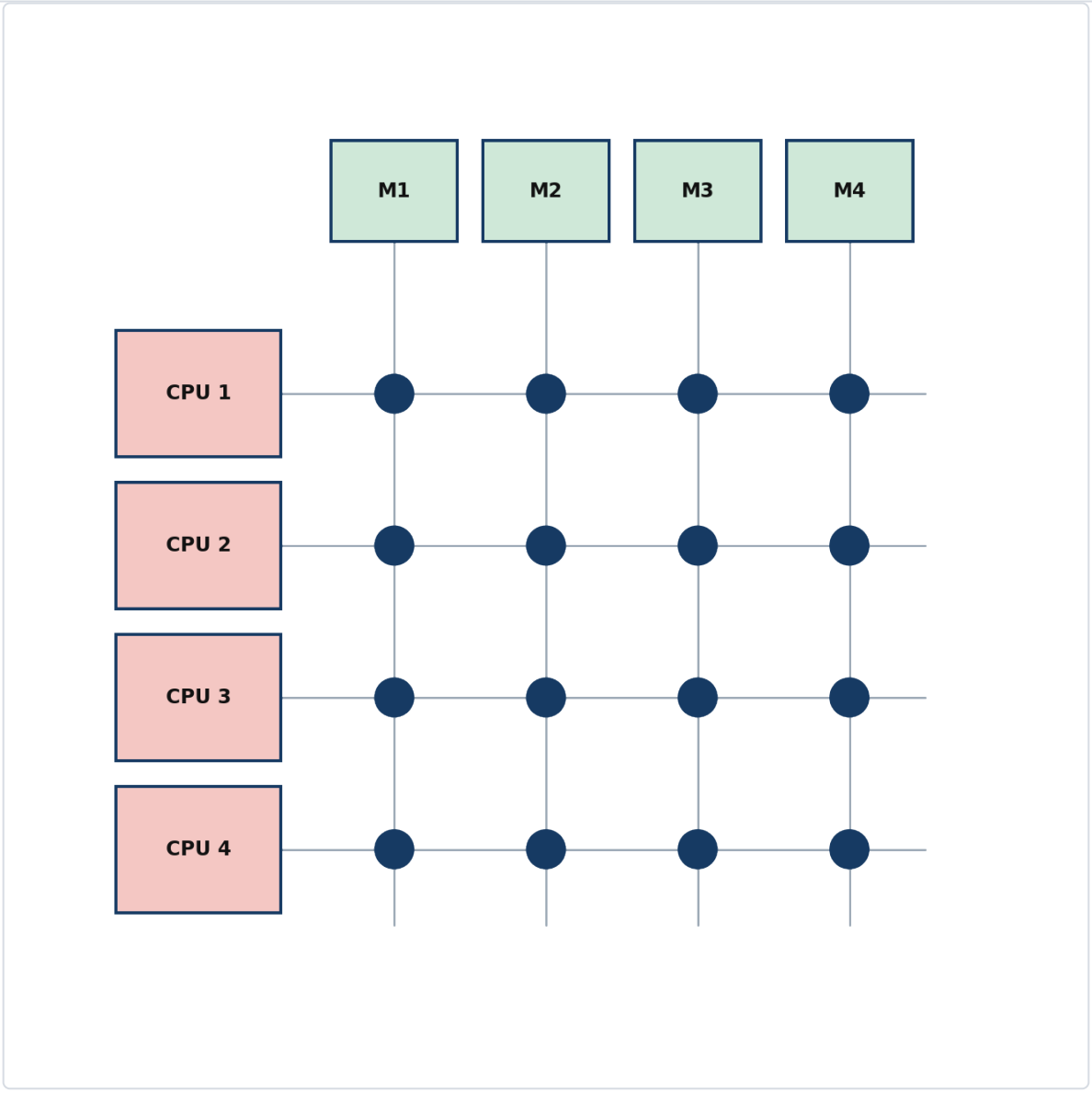


Multiport memory: every CPU has a dedicated point-to-point connection into every memory module.

Advantages	Disadvantages
Multiple CPU–memory transfers can proceed at the same time, since each path is independent.	Requires far more wiring and control logic than a shared bus — every memory module needs one port per processor.
No single shared bus to bottleneck the whole system.	Hardware cost grows quickly as more processors or memory modules are added, since connections scale with their product.

Crossbar Switch

A crossbar switch places a switching element (a "cross-point") at every intersection of a processor's row and a memory module's column, forming a full grid. Any processor can be connected to any memory module simply by closing the appropriate cross-point, and — unlike the shared bus — many such connections can be active simultaneously, as long as no two processors are trying to reach the same memory module at once.



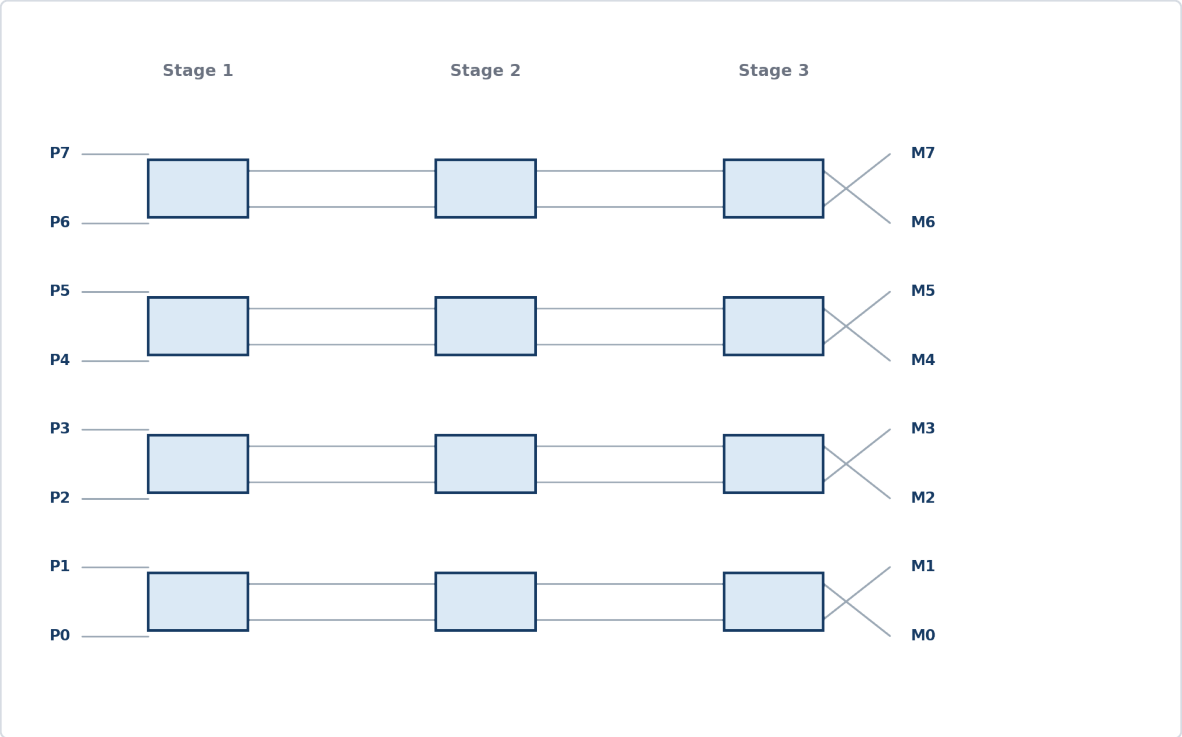
Crossbar switch: a cross-point switch sits at every processor/memory intersection, allowing many simultaneous non-conflicting connections.

Advantages	Disadvantages
Highest possible bandwidth for a given number of processors and memory modules — every non-conflicting pair can transfer data concurrently.	The number of cross-point switches grows as the *product* of the number of processors and memory modules, making it very expensive to scale to large systems.
Conceptually simple: one switch per intersection.	Physical layout and wiring become impractical once the system grows beyond a modest size.

Multistage Switching Networks

A multistage switching network builds a path between processors and memory modules out of several stages of small switching elements — commonly simple 2×2 switches, each of which can either pass its two inputs

straight through or exchange (cross) them. By chaining stages of these small switches together, a connection from any input to any output can be established without needing a dedicated cross-point for every possible pair, as a full crossbar would.

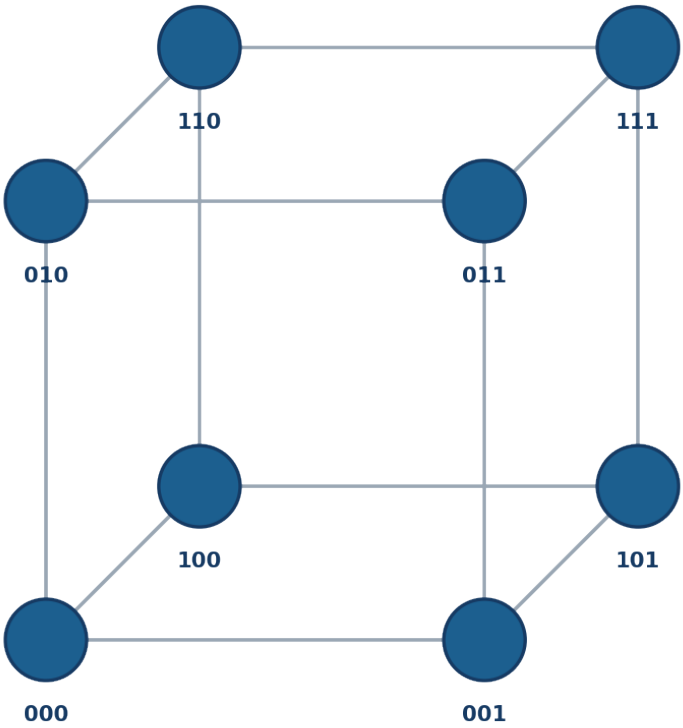


A multistage switching network: several stages of small 2×2 switches route any input to any output through a sequence of switching decisions.

Advantages	Disadvantages
Needs far fewer switching elements than a full crossbar, so it scales to much larger systems more economically.	Not every pair of processor/memory connections can be active simultaneously — some request patterns "block," meaning a path is unavailable because it conflicts with another connection already using part of the same route.
A good middle ground between the cheap-but-limited bus and the fast-but-expensive crossbar.	A request may need to pass through several stages, adding latency compared to a single-hop crossbar connection.

Hypercube Structure

In a hypercube (or binary n-cube) interconnection, each processor occupies one corner (node) of an n-dimensional cube and is given a unique n-bit binary address. Two processors are directly connected by a communication link if, and only if, their binary addresses differ in exactly one bit position. Each of the n dimensions therefore contributes exactly one direct link per node, so every node has exactly n direct neighbours in an n-cube of 2^n total nodes.



A 3-cube (n = 3): eight processors, each labelled with a 3-bit address, with a direct link between any two nodes whose addresses differ in exactly one bit.

WORKED EXAMPLE — 3-CUBE

With $n = 3$, there are $2^3 = 8$ processors, addressed 000 through 111. Node 010 connects directly to 000, 011, and 110 — each differing from 010 in exactly one bit — but not directly to 101, which differs in all three bits and must be reached by routing through at least two intermediate links.

Advantages	Disadvantages
Any node can reach any other node in at most n hops (for 2^n nodes), giving good worst-case latency even for a large processor count.	Each additional dimension doubles the number of processors, so system size is restricted to powers of two.

Regular, symmetric structure with no single central bottleneck — routing algorithms can exploit the fact that every node looks structurally identical.	Each processor needs exactly *n* physical links, so per-node wiring complexity grows as the system scales to more dimensions.

Comparing the Interconnection Schemes

Scheme	Simultaneous transfers	Hardware cost	Typical scale
Time-shared common bus	One at a time	Lowest	Small systems
Multiport memory	One per port, per module	Grows with P × M	Small–medium systems
Crossbar switch	Many, non-conflicting pairs	Highest — grows with P × M	Small–medium, high-performance systems
Multistage network	Many, subject to blocking	Moderate — grows with P·log(P)	Medium–large systems
Hypercube	Many, via multi-hop routing	Moderate — n links per node	Large, distributed-memory systems

P = number of processors, M = number of memory modules.

COURSE LECTURE NOTES

Computer Organization & Architecture

UNIT V: Input-Output Organization

CORE SYLLABUS COVERAGE

- Peripheral Devices & Bus Mismatches
- Input-Output Interface Context
- Asynchronous Data Transfer Protocols
- Strobe Control Mechanism (Source & Destination)
- Handshaking Protocol (Source & Destination)
- Asynchronous Serial Transfer & Bit Framing

Department of Computer Engineering

State Board of Technical Education

1. Input-Output (I/O) Interface

In a computer system, the Central Processing Unit (CPU) and core Memory form the computational heart, while peripheral devices (such as keyboards, printers, magnetic disks, and displays) allow interaction with the external world. However, these external peripherals cannot be linked directly to the system bus.

> **Definition: I/O Interface**
>
> An **I/O Interface** is a specialized electronic circuit or hardware component placed between the system bus and a peripheral device. It resolves all architectural, signal, and timing mismatches, supervising the smooth transfer of data between the internal bus system and the connected external device.

Reasons for Requiring an I/O Interface

Peripherals vary widely in their physical structure, electrical properties, and operation speeds. Direct attachment to the CPU bus is impossible due to the following structural mismatches:

- **Operating Speed Mismatch:** Peripherals are electromagnetic or electromechanical devices that operate at drastically slower speeds than the ultra-fast, purely electronic CPU and main memory. Direct connection would result in severe CPU starvation or bus stalling.
- **Signal Level Mismatch:** The CPU operates on highly controlled digital logic levels (e.g., standard TTL/CMOS voltages). Peripherals may employ completely different operating voltages, analog waveforms, or mechanical pulses.
- **Data Format & Word Length Mismatch:** Internal system buses transfer data in parallel words (e.g., 32-bit or 64-bit blocks). Many peripherals transmit data serially (one bit at a time) or utilize distinct encoding schemes and character word lengths.

2. Asynchronous Data Transfer Modes

Data transmission between two separate independent devices within a computer system can be coordinated via two fundamental methods based on timing synchronization:

Synchronous Data Transfer

In a synchronous architecture, both the transmitting unit and the receiving unit share a **common clock signal** line. All data movements, bus sampling, and internal operations are rigidly bound to the rising or falling edges of this system-wide master clock. While structurally high-speed, it demands that all devices operate at perfectly uniform, predictable timings.

Asynchronous Data Transfer

In an asynchronous architecture, the internal timing domains of the transmitter and receiver are entirely independent. They **do not share a common clock signal**. Because the units operate at different rates, specific control signals must be generated dynamically to inform the receiver that data is ready, or to inform the transmitter that the receiver is ready to accept data.

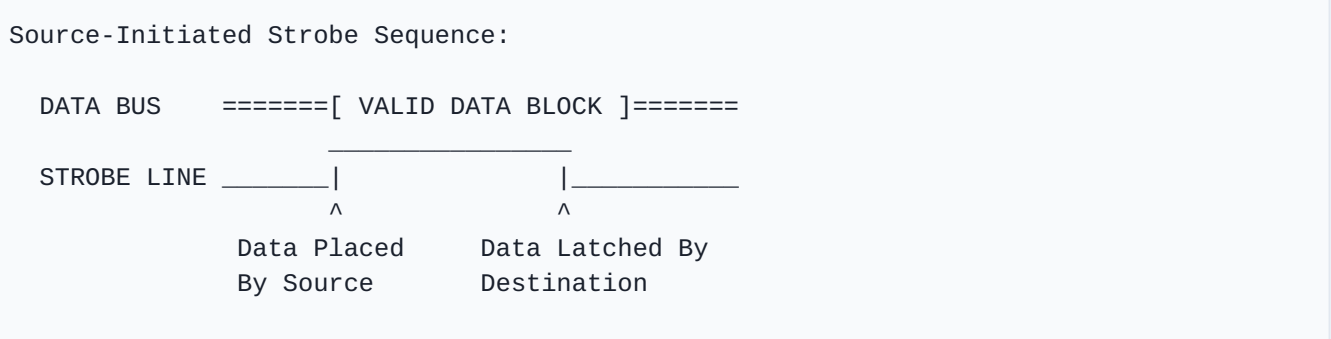
To implement Asynchronous Data Transfer, two dominant hardware protocols are used: **Strobe Control** and **Handshaking**.

A. Strobe Control Method

The Strobe Control method relies on a single control line—termed the **Strobe**—to coordinate the exact moment of data capture. The strobe pulse can be driven by either the sending device (Source) or the receiving device (Destination).

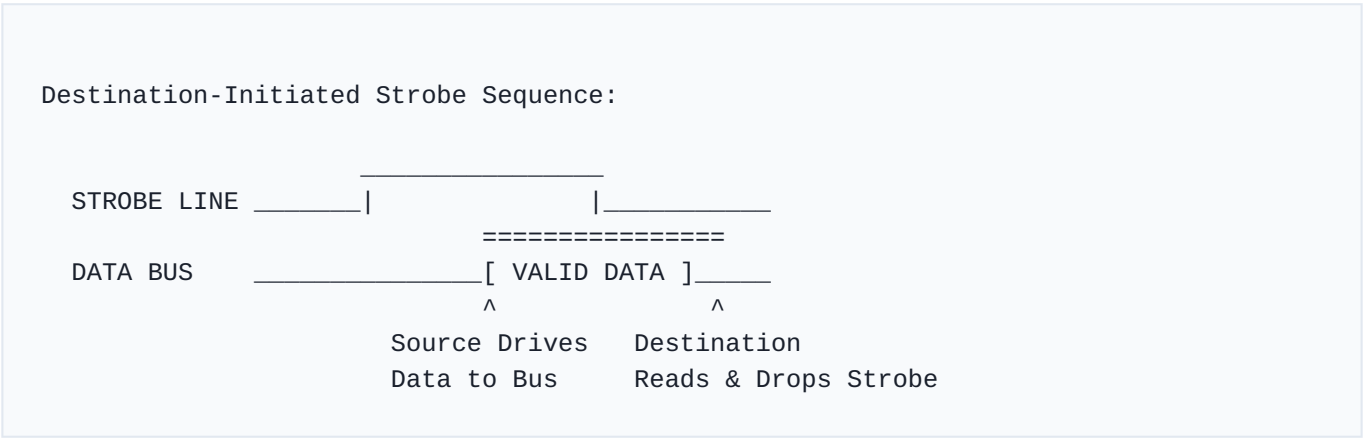
I. Source-Initiated Strobe

The transferring unit (Source) places the data on the shared data bus. After allowing time for the electrical signals to settle on the bus wires, the Source activates the Strobe signal. The destination monitors this line, perceives the transition, and latches the data. After a predetermined brief interval, both the strobe signal and the data are cleared from the bus.



II. Destination-Initiated Strobe

The receiving unit (Destination) initiates the cycle by pulling the Strobe line active, effectively requesting data from the Source. The Source detects this active line, retrieves the target data, and drives it onto the data bus. The Destination reads the data off the bus and then terminates the strobe signal. The Source then safely clears the data bus.



> **Critical Vulnerability of Strobe Control**
>
> The primary deficiency of the Strobe method is its **lack of confirmation (No Acknowledgment)**. In a source-initiated strobe, the sender assumes the receiver read the data but has no true verification. In a destination-initiated strobe, the receiver assumes the sender placed valid data, but cannot dynamically confirm if the sender failed or stalled.

B. Handshaking Protocol

The Handshaking protocol remedies the limitations of strobe control by introducing a **second, independent control line** tasked exclusively with providing a formal acknowledgment. This establishes a true closed-loop, two-way hardware agreement before data is modified or cleared.

I. Source-Initiated Handshaking

This implementation utilizes two critical control lines:

- `Data Ready`: Driven by the Source to indicate valid data is present on the bus.
- `Data Accepted`: Driven by the Destination to confirm successful ingestion of the data word.

Step	Source Action	Destination Action
Step 1	Places valid data on the bus and activates `Data Ready`.	Senses `Data Ready` active, prepares to process.
Step 2	Maintains data on the bus, waits for response.	Reads the data lines and activates `Data Accepted`.
Step 3	Detects `Data Accepted`, deactivates `Data Ready`, clears data.	Senses `Data Ready` drop, remains active.
Step 4	Waits for acknowledgment line to clear.	Deactivates `Data Accepted`; line returns to idle.

II. Destination-Initiated Handshaking

This implementation also employs two distinct lines operating in reverse logical order:

- `Ready for Data`: Driven by the Destination indicating it is free to accept a new data word.
- `Data Valid`: Driven by the Source once it responds by writing true data onto the bus.

```
Destination-Initiated Handshaking Flow:

  1. Destination asserts [Ready for Data]
  2. Source senses line -> Places Data on Bus -> Asserts [Data Valid]
  3. Destination senses line -> Reads Bus Data -> De-asserts [Ready for Data]
  4. Source senses line drop -> Removes Data -> De-asserts [Data Valid]
```

3. Asynchronous Serial Transfer

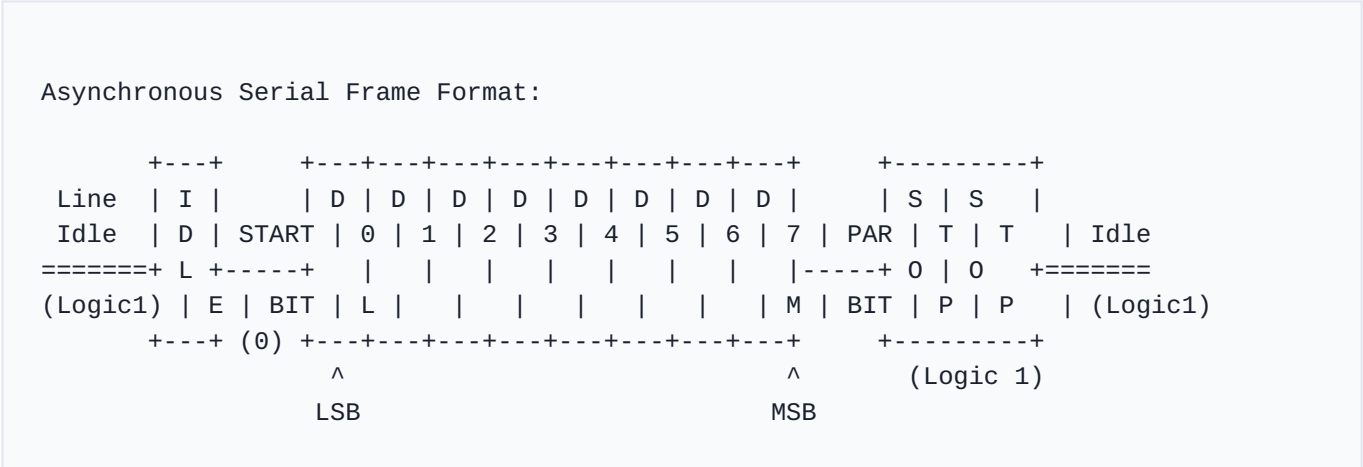
In a **Serial Transfer** scheme, data words are broken down and transmitted sequentially **one single bit at a time** over a solitary communication channel or wire. This stands in contrast to parallel structures where multiple lines carry bits simultaneously. When executed asynchronously, the receiving hardware does not share a master clock with the transmitter.

To achieve synchronization and prevent bit-shifting errors, the transmission protocol relies on precise, uniform bit durations and a rigid protocol known as **Character Framing**.

The Asynchronous Bit-Frame Structure

Every isolated character (typically an 8-bit block or ASCII character) is wrapped in structural bits that dictate the timing bounds:

- **Idle State (Mark):** When no active communication is taking place, the serial line is continuously maintained at a high binary logic state (Logic `1`).
- **Start Bit (Space):** The beginning of a new character transmission is signaled when the transmitter forces the line down to a low binary state (Logic `0`) for exactly one bit-duration. This abrupt drop alerts the receiver's internal high-speed sampling clock to begin counting cycles.
- **Data Bits:** Immediately following the Start Bit, the information bits composing the actual character are driven onto the line, starting invariably with the Least Significant Bit (LSB) and ending with the Most Significant Bit (MSB). Typically spans 5 to 8 bits.
- **Parity Bit:** An optional single bit appended immediately after the payload for elementary error detection (configured as either Even or Odd parity checking).
- **Stop Bit(s):** To conclude the character sequence, the transmitter pulls the line back to the high state (Logic `1`) for a minimum standard duration (1, 1.5, or 2 bit periods). This deliberate return to the high state forces an idle interval, allowing the receiver to finalize internal buffering and wait reliably for the next low-going Start Bit.



COMPUTER ORGANIZATION

Question Bank Collection

Course: 4th Semester, Computer Engineering / CSE / IT **Time Allowed:** 3 Hours

Subject Code: 180845 / 170845 / 120845

📑 PAPER SET 1

SECTION – A

Note: Multiple Choice Questions. All questions are compulsory. (10 × 1 = 10 Marks)

Q.1 RISC stands for: **(CO3)**

a) Reduced instruction set compiler
b) Reduced instruction set computer
c) Random instruction set compiler
d) None of them

Q.2 CISC stands for: **(CO3)**

a) Complex instruction set compiler
b) Compiled instruction set computer
c) Complex instruction set computer
d) None of them

Q.3 RAM is also called: **(CO4)**

a) Read access memory
b) Read/write memory
c) Read only memory
d) None of them

Q.4 Which memory device is made of semiconductor? **(CO4)**

a) RAM
b) Hard Disk
c) Floppy disk
d) CD disk

Q.5 _____ bus structure is usually used to connect I/O devices. **(CO5)**

a) Multiple bus
b) Single bus
c) Star bus
d) Rambus

Q.6 During the transfer of data between the processor and memory, we use: **(CO4)**

a) Cache
b) TLB
c) Buffers
d) Register

Q.7 The RISC processor has a more complicated design than CISC. (True / False) **(CO3)**

Q.8 Which architecture is power efficient? **(CO3)**

a) CISC
b) RISC
c) ISA
d) IANA

Q.9 BIOS means: (CO3)

a) Bidirectional input output system
b) Basic interfacing output system
c) Basic Input Output System
d) None of them

Q.10 DMA stands for: (CO3)

a) Digital Memory Access
b) Direct memory address
c) Direct Memory Access
d) None of them

SECTION – B

Note: Objective Completion / Fill-in-the-blanks type questions. All questions are compulsory. (10 × 1 = 10 Marks)

Q.11 CPU stands for _____. (CO1)

Q.12 Expand EPROM. (CO4)

Q.13 CD-ROM stands for _____. (CO4)

Q.14 Define immediate mode. (CO5)

Q.15 Expand SIMD. (CO6)

Q.16 Expand MIMD. (CO6)

Q.17 Name two types of stack. (CO1)

Q.18 ALU performs the required micro-operations for executing the _____. (CO1)

Q.19 The I/O bus consists of _____, _____, and _____. (CO5)

Q.20 _____ is a technique of decomposing a sequential process into sub-operations. (CO6)

SECTION – C

Note: Short Answer type questions. Attempt any twelve questions out of fifteen. (12 × 5 = 60 Marks)

Q.21 What do you mean by instruction format? (CO1)

Q.22 Differentiate between RISC and CISC processors. (CO3)

Q.23 What is a stack? Explain its types in brief. (CO1)

Q.24 What is cache memory? (CO4)

Q.25 Explain virtual memory. (CO4)

Q.26 What do you mean by the term Memory? (CO4)

Q.27 Explain the input and output interface. (CO5)

Q.28 Explain DMA (Direct Memory Access). (CO6)

Q.29 Explain FIFO buffer. **(CO5)**

Q.30 What are optical memories? **(CO4)**

Q.31 What is a magnetic disk? **(CO4)**

Q.32 Explain Control Word. **(CO2)**

Q.33 Write a short note on multiprocessor organization. **(CO6)**

Q.34 What are the steps needed to process an instruction? **(CO6)**

Q.35 What is parallel processing? **(CO6)**

SECTION – D

Note: Long Answer type questions. Attempt any two questions out of three. (2 × 10 = 20 Marks)

Q.36 What is a CPU? Explain the general register organization in detail. **(CO1)**

Q.37 Explain the memory hierarchy in detail. **(CO4)**

Q.38 Explain the various types of pipelining. **(CO6)**

📑 PAPER SET 2

SECTION – A

Note: Multiple Choice Questions. All questions are compulsory. (10 × 1 = 10 Marks)

Q.1 Input device is: **(CO1)**

a) Keyboard
b) Printer
c) Monitor
d) Plotter

Q.2 RISC stands for: **(CO1)**

a) Reduced Instruction Set Computer
b) Read Instruction Set Computer
c) Reduced Instruction Set Coming
d) Reduced Input Self Computer

Q.3 1 GB = _____ Bytes. **(CO1)**

a) 1,000
b) 1,000,000,000,000
c) 1,000,000
d) 1,000,000,000

Q.4 RAM is: **(CO2)**

a) Volatile memory
b) Static Memory
c) Garbage memory
d) Low speed memory

Q.5 An address generated by the CPU is generally referred to as: **(CO2)**

a) Physical Address
b) Associative Address
c) Referral Address
d) Logical Address

Q.6 Which of the following is not a type of ROM? **(CO3)**

a) PROM
b) EEPROM
c) EAROM
d) MEPROM

Q.7 RAM can be: **(CO3)**

a) SRAM, DRAM
b) ROM
c) PROM
d) MEPROM

Q.8 A _____ buffer can be used for the fetch segment. **(CO4)**

a) MIFO
b) SIFO
c) FIFO
d) LIFO

Q.9 Name the parallel processing architecture types: **(CO-)**

a) SIMD, MIMD
b) MISD
c) SISD
d) All of the above

Q.10 Parallel processor is associated with: **(CO4)**

a) Distributed architecture
b) Pipelining
c) BIOS
d) RISC

SECTION – B

Note: Objective / Short completion questions. All questions are compulsory. (10 × 1 = 10 Marks)

Q.11 The _____ stores intermediate data used during the execution of instructions. **(CO1)**

Q.12 ALU performs micro-operations for executing the _____. **(CO1)**

Q.13 The register that holds the address for the stack is called _____. **(CO1)**

Q.14 The _____ points at the address of the next instruction in the program. **(CO1)**

Q.15 Memory refers to the _____ of a computer system. **(CO2)**

Q.16 Parts of primary memory are _____ and _____. **(CO2)**

Q.17 EPROM stands for _____. **(CO2)**

Q.18 Access time = _____. **(CO2)**

Q.19 I/O Bus consists of _____ and _____. **(CO3)**

Q.20 In parallel MIMD systems, _____ communication is essential for processing. **(CO4)**

SECTION – C

Note: Short Answer type questions. Attempt any twelve questions out of fifteen. (12 × 5 = 60 Marks)

Q.21 Explain One-Address Instructions. **(CO1)**

Q.22 Explain: 1) Direct Addressing Mode, 2) Indirect Addressing Mode. **(CO1)**

Q.23 What are the steps followed by the CPU when an interrupt occurs? **(CO1)**

Q.24 Compare internal interrupts and external interrupts. **(CO1)**

Q.25 Write a short note on: 1) SRAM, 2) DRAM. **(CO2)**

Q.26 Why is virtual memory used in computer systems? **(CO2)**

Q.27 What is memory mapping? Explain. **(CO2)**

Q.28 What are the components of a memory management unit (MMU)? **(CO2)**

Q.29 What is the difference between static RAM and dynamic RAM? **(CO2)**

Q.30 What are the major functions of BIOS? **(CO3)**

Q.31 Write a short note on synchronous and asynchronous data transfer. **(CO3)**

Q.32 Explain briefly the interrupt priority encoder. **(CO3)**

Q.33 Explain the types of parallel processing. **(CO4)**

Q.34 What is Reverse Polish Notation (RPN)? **(CO4)**

Q.35 What are the various characteristics of multiprocessors? **(CO4)**

SECTION – D

Note: Long Answer type questions. Attempt any two questions out of three. (2 × 10 = 20 Marks)

Q.36 Explain the characteristics of RISC architecture. **(CO1)**

Q.37 Write short notes on: **(CO2)**

1. Virtual Memory 2. Demand Paging 3. Associative Memory

Q.38 Answer the following: **(CO4)**

1. Explain various types of pipelining.
2. Discuss the characteristics of computer architecture.

📑 PAPER SET 3

SECTION – A

Note: Objective type questions. All questions are compulsory. (10 × 1 = 10 Marks)

Q.1 RISC stands for _____. **(CO1)**

Q.2 One byte is equivalent to _____ bits. **(CO1)**

Q.3 ANSI stands for _____. **(CO1)**

Q.4 Classify computers according to Flynn's classification. **(CO4)**

Q.5 The full form of WORM is _____. **(CO2)**

Q.6 BIOS means _____. **(CO2)**

Q.7 Full form of DMA is _____. **(CO3)**

Q.8 CMOS stands for _____. **(CO3)**

Q.9 Hardwired control units feature a rigid application environment. (True / False) **(CO1)**

Q.10 SISD stands for _____. **(CO4)**

SECTION – B

Note: Very Short Answer type questions. Attempt any ten questions. (10 × 2 = 20 Marks)

Q.11 What is the function of the Control Unit? **(CO3)**

Q.12 Explain Bootstrap Loader. **(CO3)**

Q.13 Name various types of parallel processors. **(CO4)**

Q.14 Explain DMA Transfer. **(CO3)**

Q.15 Define POST (Power-On Self-Test). **(CO3)**

Q.16 Draw a block diagram of Associative Memory. **(CO2)**

Q.17 Define RAM and ROM. **(CO2)**

Q.18 Write a short note on Register Indirect Mode. **(CO2)**

Q.19 Draw a block diagram of segmentation with paging. **(CO3)**

Q.20 State the principle of Single Accumulator Organization. **(CO1)**

Q.21 Define three-address instructions. **(CO1)**

Q.22 Define Auto-Increment Mode. **(CO1)**

SECTION – C

Note: Short Answer type questions. Attempt any eight questions. (8 × 5 = 40 Marks)

Q.23 Write a short note on Direct Memory Access (DMA). **(CO3)**

Q.24 Explain Arithmetic Pipelining. **(CO4)**

Q.25 Differentiate between RISC and CISC. **(CO1)**

Q.26 What are priority interrupts? **(CO3)**

Q.27 Explain memory hierarchy. **(CO2)**

Q.28 State and explain BIOS. **(CO3)**

Q.29 Write a note on Cross-Bar Switches. **(CO4)**

Q.30 What is the basic concept of Virtual and Cache Memory? **(CO2)**

Q.31 Write short notes on: **(CO2)**

a) Hit rate b) DVD

Q.32 Differentiate between Access Time and Latency Time. **(CO2)**

SECTION – D

Note: Long Answer type questions. Attempt any three questions. (3 × 10 = 30 Marks)

Q.33 What is an addressing mode? Explain the different types of addressing modes. **(CO1)**

Q.34 Explain pipelining and its techniques using a clean diagram. **(CO4)**

Q.35 Differentiate between Direct Mapping, Associative Mapping, and Set-Associative Mapping. **(CO2)**

Q.36 Write short notes on: **(CO4/CO3)**

a) Multi-Processing

b) Functions of BIOS