MODULE 1

1. Input–Output Organization

Input–Output (I/O) organization deals with the methods and hardware used to transfer data between the CPU/memory and external peripheral devices such as keyboards, displays, disks, and network adapters. Because peripherals operate at widely different speeds and use different data formats compared to the CPU, specialized interface hardware and transfer techniques are required.

1.1 Input-Output Interface

An I/O interface provides a method for transferring information between internal storage (registers, memory) and external I/O devices. It resolves the differences between the CPU and peripherals in terms of signal types, data formats, and operating speeds.

- Peripherals are typically slower, electromechanical devices, while the CPU and memory operate electronically at much higher speed — the interface synchronizes this speed mismatch.
- Data formats and word lengths of peripherals often differ from the CPU's internal word length, so the interface performs format conversion.
- The interface typically contains data registers, status registers, and control registers/logic, accessible by the CPU through I/O commands.

1.1.1 I/O Bus and Interface Registers

The I/O bus consists of data lines, address lines, and control lines (read/write/select) that connect the CPU to multiple I/O interface units. Each interface unit is assigned a unique address so that the CPU can select a specific device and register when issuing I/O commands.

Register	Purpose
Data Register	Holds the data byte/word being transferred to or from the device.
Status Register	Holds flag bits (e.g., busy, ready, error) that the CPU can test before/after a transfer.
Control Register	Receives commands from the CPU that control the operating mode of the device.

1.1.2 I/O versus Memory Addressing

- Isolated (separate) I/O: I/O devices have their own distinct address space, accessed by dedicated I/O instructions (e.g., IN, OUT).
- Memory-mapped I/O: I/O device registers share the same address space as memory, so the same instructions used to access memory can also access I/O devices.

1.2 Asynchronous Data Transfer

Asynchronous data transfer is used between two independent units (e.g., CPU and peripheral) that do not share a common clock, requiring an explicit handshaking mechanism to coordinate the timing of each transfer.

1.2.1 Strobe Control

A single control line (strobe) is pulsed by the source or destination to indicate that valid data is available on the data bus, or that data has been accepted. This is simple but provides no confirmation that the transfer was actually completed by the receiving unit.

1.2.2 Handshaking

Handshaking uses two control lines to provide a more reliable transfer: one signal (e.g., 'data valid') is generated by the source, and a second signal (e.g., 'data accepted') is generated by the destination, with each unit's action triggered by a change in the other's signal.

- Source-initiated handshaking: the source asserts 'data valid'; the destination responds with 'data accepted' after reading the data.
- Destination-initiated handshaking: the destination asserts a 'ready for data' signal; the source then places data on the bus and asserts 'data valid'.

Note: Handshaking guarantees correct transfer timing regardless of variations in device speed, since each side waits for explicit confirmation from the other before proceeding.

1.3 Modes of Transfer

Binary data transfer between the CPU and I/O devices can be handled in several ways, differing in the amount of CPU involvement and the transfer speed achieved.

Mode of Transfer	Description
Programmed I/O	The CPU executes a program that directly controls the transfer, repeatedly polling the status register (busy-waiting) until the device is ready. Simple to implement, but wastes CPU time.
Interrupt-Initiated I/O	The CPU issues a transfer request and continues executing other tasks; the device interrupts the CPU only when it is ready, avoiding busy-waiting.
Direct Memory Access (DMA)	A dedicated DMA controller transfers data directly between the I/O device and memory, without CPU involvement in the actual transfer — used for high-speed bulk transfers.

1.4 Priority Interrupt

When multiple devices can generate interrupt requests, a priority interrupt system is used to determine which device's request should be serviced first when several requests occur simultaneously, and to establish the order in which pending requests are handled.

1.4.1 Software-Implemented Priority (Polling)

When an interrupt occurs, the CPU executes a polling routine that checks each device's status flag, in a fixed priority order, to identify which device requested service. This is simple but slow, since the polling time increases with the number of devices.

1.4.2 Hardware-Implemented Priority

- Daisy-Chain Priority: Devices are connected in series; a device with a pending interrupt blocks the acknowledge signal from reaching lower-priority devices, giving a simple, low-cost hardware priority scheme based on physical position in the chain.
- Parallel Priority (Priority Encoder): Each device has its own interrupt request line, and a priority encoder circuit generates a code identifying the highest-priority active request, giving faster response than daisy-chaining.

1.4.3 Interrupt Cycle

On recognizing an interrupt request, the CPU typically completes the current instruction, saves the current program counter (and often status flags) onto the stack, and branches to the appropriate interrupt-service routine; upon completion, a return-from-interrupt instruction restores the saved state and resumes the interrupted program.

1.5 Direct Memory Access (DMA)

Direct Memory Access allows an I/O device to transfer blocks of data directly to or from memory without the data passing through CPU registers, greatly increasing transfer speed for high-volume I/O devices such as disks.

1.5.1 DMA Controller

A DMA controller takes over control of the system buses from the CPU for the duration of the data transfer. It contains an address register (holding the current memory address), a word-count register (tracking the number of words remaining to transfer), and control logic to manage the transfer and signal its completion.

1.5.2 DMA Transfer Procedure

1. 1. The CPU initializes the DMA controller by loading the starting memory address, the word count, and the transfer direction (read/write).
2. 2. The CPU issues a start command and continues executing its own program.
3. 3. The DMA controller requests control of the system bus (via a bus request/bus grant handshake) whenever the I/O device is ready to transfer a word.
4. 4. Once granted the bus, the DMA controller transfers data directly between the device and memory, incrementing its address register and decrementing its word-count register after each word.
5. 5. When the word count reaches zero, the DMA controller releases the bus and typically interrupts the CPU to signal that the block transfer is complete.

1.5.3 DMA Bus Arbitration Modes

- Burst (Block) Transfer: The DMA controller retains control of the bus until the entire data block has been transferred.
- Cycle Stealing: The DMA controller transfers one word at a time, then releases the bus back to the CPU between transfers, 'stealing' individual bus cycles rather than blocking the CPU entirely.

Note: DMA dramatically reduces CPU overhead for large data transfers, since the CPU only needs to initialize the transfer and respond to a single completion interrupt, rather than managing every word individually.

MODULE 2

2. Memory Organization

Memory organization addresses how a computer system structures its storage resources to balance speed, capacity, and cost. No single memory technology can simultaneously provide very high speed, very large capacity, and very low cost, so practical computer systems use a hierarchy of memory types, each with different characteristics.

2.1 Memory Hierarchy

The memory hierarchy arranges storage devices in layers, from the fastest and most expensive (but smallest) at the top, to the slowest and cheapest (but largest) at the bottom. Data that is frequently accessed tends to migrate toward the faster levels, exploiting the principle of locality of reference.

Level	Typical Technology	Relative Speed	Relative Size / Cost per bit
Registers	Flip-flops (inside CPU)	Fastest	Smallest / Highest cost
Cache Memory	SRAM	Very fast	Small / High cost
Main Memory	DRAM	Fast	Medium / Medium cost
Auxiliary (Secondary) Memory	Magnetic disk, SSD, optical media	Slow	Large / Low cost
Off-line / Archival Storage	Magnetic tape, cloud/archival storage	Slowest	Largest / Lowest cost

Note: The goal of the memory hierarchy is to give the CPU the illusion of a memory that is as fast as the fastest level and as large and cheap as the slowest level, by keeping frequently used data in the faster levels.

2.2 Main Memory

Main memory (primary memory) is the memory unit that communicates directly with the CPU, holding the programs and data currently in use. It is typically implemented using semiconductor RAM (Random Access Memory) technology.

2.2.1 RAM and ROM

- RAM (Random Access Memory): Volatile memory that can be both read from and written to; loses its contents when power is removed. Common types include Static RAM (SRAM), which stores bits in flip-flops and needs no refreshing, and Dynamic RAM (DRAM), which stores bits as charge on capacitors and requires periodic refreshing.
- ROM (Read-Only Memory): Non-volatile memory that retains its contents without power; used mainly to store fixed programs (e.g., bootstrap loaders/firmware) that do not change during normal operation. Variants include PROM, EPROM, and EEPROM, which allow the stored contents to be programmed (and in some cases erased and reprogrammed).

2.2.2 Memory Address Map

A memory address map is a table (or diagram) showing the range of addresses assigned to each memory chip/unit in the system, used during system design to allocate the address space among RAM, ROM, and I/O devices (in memory-mapped I/O systems).

2.2.3 Memory Connection to CPU

Main memory connects to the CPU through address lines (selecting the memory word), data lines (carrying the word being read or written), and control lines (such as Read and Write) that specify the type of memory operation being performed.

2.3 Auxiliary Memory

Auxiliary (secondary) memory provides large-capacity, non-volatile storage for programs and data that are not currently being processed by the CPU, at a much lower cost per bit than main memory, though with significantly slower access time.

- Magnetic Disks: Data is recorded on concentric tracks (further divided into sectors) on a rotating magnetic surface; a read/write head accesses data by moving to the correct track and waiting for the desired sector to rotate underneath it. Widely used as the primary secondary-storage device in many systems.
- Magnetic Tape: A sequential-access storage medium, historically used for backup and archival storage, where data must be read in order rather than being directly addressed.
- Solid-State Drives (SSDs) / Flash Memory: Non-volatile semiconductor storage with much faster access than magnetic disks and no moving parts, increasingly used in place of traditional magnetic disks.
- Optical Disks (CD/DVD/Blu-ray): Store data using patterns read by a laser; commonly used for distribution media and archival storage.

Key performance parameters of auxiliary memory include access time (seek time + rotational latency, for disks), transfer rate, and storage capacity.

2.4 Associative Memory

Associative memory, also called Content-Addressable Memory (CAM), is a memory device in which stored data is accessed by its content rather than by its address — the memory is searched in parallel for a match against a given search argument, and the associated data is retrieved from any location where a match is found.

2.4.1 Structure and Operation

- An argument register holds the value being searched for.
- A key (mask) register specifies which bits of the argument are actually to be used in the comparison, allowing partial-field matching.
- Each word in the associative memory is compared, in parallel, against the argument register (through the mask); a match register indicates which word(s) matched.

Note: Because comparison is performed in parallel across all stored words simultaneously, associative memory offers extremely fast search operations, but at significantly higher hardware cost than conventional address-based memory.

2.4.2 Applications

- Cache memory tag comparison (in fully associative and set-associative cache designs).
- Database search and retrieval operations requiring rapid content-based lookup.
- Address translation tables in virtual memory systems and network routing table lookups.

2.5 Cache Memory

Cache memory is a small, very fast memory placed between the CPU and main memory, used to hold copies of the most recently (or most frequently) used instructions and data, reducing the average memory access time seen by the CPU.

2.5.1 Principle of Locality

- Temporal locality: recently accessed memory locations are likely to be accessed again in the near future.
- Spatial locality: memory locations near a recently accessed location are likely to be accessed soon as well.

Cache memory exploits both forms of locality by transferring blocks of nearby data into cache whenever a memory location is accessed.

2.5.2 Cache Hit and Miss

- Cache hit: the CPU finds the requested data already in cache, allowing a fast access.
- Cache miss: the requested data is not in cache, requiring a slower access to main memory (and typically loading the corresponding block into cache for future use).

The hit ratio (fraction of accesses that are hits) is the key performance metric of a cache; a high hit ratio allows the system to achieve an average access time close to that of the cache itself, despite the much larger and slower main memory behind it.

2.5.3 Cache Mapping Techniques

Mapping Technique	Description
Direct Mapping	Each main memory block maps to exactly one specific cache line, determined by (block address) mod (number of cache lines); simple and fast, but prone to conflict misses when multiple frequently used blocks map to the same line.
Associative (Fully Associative) Mapping	A main memory block can be placed in any cache line; requires an associative (content-addressable) search of all cache tags to locate data, giving maximum flexibility at higher hardware cost.
Set-Associative Mapping	A compromise: cache lines are grouped into sets, and a given memory block can be placed in any line within its designated set; commonly used in practice (e.g., 2-way, 4-way set-associative caches).

2.5.4 Cache Write Policies

- Write-through: every write to cache is immediately also written to main memory, keeping memory always up to date at the cost of extra write traffic.
- Write-back: writes update only the cache initially (marking the line as 'dirty'); the modified data is written to main memory only when that cache line is later replaced, reducing memory traffic.

2.5.5 Multilevel Cache

Modern processors commonly use multiple cache levels (L1, L2, and often L3), with L1 being the smallest and fastest (closest to the CPU core) and each subsequent level being larger, slightly slower, and shared among more cores, further reducing the effective average memory access time.