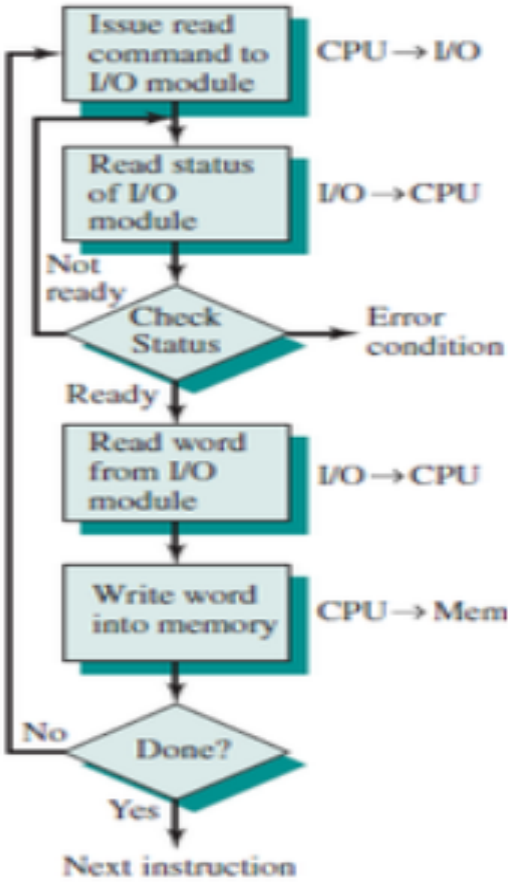
I. Techniques for performing I/O operations

References: https://inputoutput5822.weebly.com/interrupt-driven-io.html

1. Programmed I/O

- **Concept:**
  In this method, the **CPU is in complete control** of all I/O operations. The CPU issues an I/O command to a device and then **continuously checks (polls)** the device status to see if it's ready to transfer data.
- <With programmed I/O, data are exchanged between the processor and the I/O module. The processor executes a program that gives it direct control of the I/O operation, including sensing device status, sending a read or write command, and transferring the data. When the processor issues a command to the I/O module, it must wait until the I/O operation is complete. If the processor is faster than the I/O module, this is wasteful of processor time.>
- **Steps:**
  1. CPU sends an I/O command to the device.
  2. CPU waits until the device sets its status as ready.
  3. CPU transfers data between device and memory (or registers).
  4. CPU continues to the next instruction.
- **Advantages:**
  - Simple and easy to implement.
  - No hardware support required for interrupts.
- **Disadvantages:**
  - **CPU is busy waiting** (wastes time checking device status).
  - Not suitable for slow devices or multitasking systems.
- **Example:**
  Keyboard input handled by the CPU checking if a key is pressed.



For input Data Transfer:

1. Each input is read after first testing whether the device is ready with the input (a state reflected by a bit in a status register).
2. The program waits for the ready status by repeatedly testing the status bit and till all targeted bytes are read from the input device.
3. The program is in busy (non-waiting) state only after the device gets ready else in wait state.

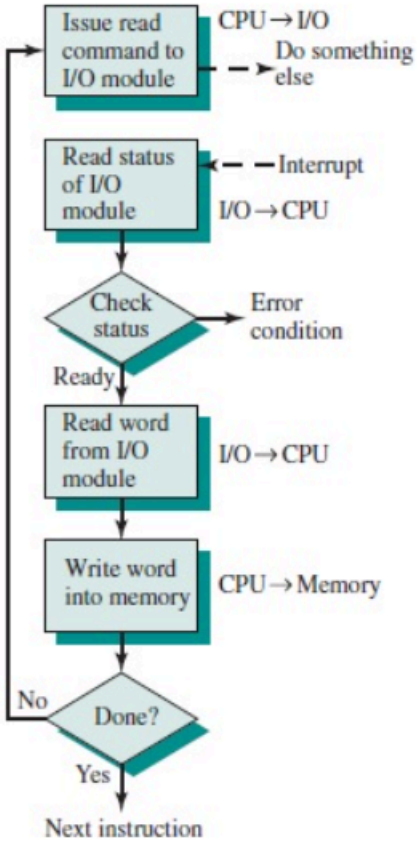
For Output data Transfer:

1. Each output written after first testing whether the device is ready to accept the byte at its output register or output buffer is empty.
2. The program waits for the ready status by repeatedly testing the status bit(s) and till all the targeted bytes are written to the device.
3. The program in busy (non-waiting) state only after the device gets ready else wait state.

---

2. Interrupt-Initiated I/O

- **Concept:**
  In this method, the **CPU does not waste time waiting** for the device. Instead, after issuing an I/O command, the CPU continues executing other instructions.
  When the device is ready, it **sends an interrupt signal** to the CPU.
- <Interrupt I/O is a way of controlling input/output activity whereby a peripheral or terminal that needs to make or receive a data transfer sends a signal. This will cause a program interrupt to be set. At a time appropriate to the priority level of the I/O interrupt. Relative to the total interrupt system, the processors enter an interrupt service routine. The function of the routine will depend upon the system of interrupt levels and priorities that is implemented in the processor. The interrupt technique requires more complex hardware and software, but makes far more efficient use of the computer's time and capacities.>
- **Steps:**
  1. CPU issues I/O command to the device.
  2. Device performs the operation independently.
  3. When ready, the device sends an **interrupt** to the CPU.
  4. CPU stops current execution, handles the interrupt (data transfer), and then resumes normal work.
- **Advantages:**
  - CPU time is used efficiently.
  - Better performance than programmed I/O.
- **Disadvantages:**
  - More complex (requires interrupt handling mechanism).
  - Frequent interrupts can affect system performance.
- **Example:**
  Printer notifying the CPU after completing a print job.



---

3. Direct Memory Access (DMA)

- **Concept:**
  In DMA, the **data transfer happens directly between the I/O device and main memory**, **bypassing the CPU**.
  A **DMA controller (DMAC)** manages the transfer.

- **Steps:**

  1. CPU initializes the DMA controller with memory address, device address, and transfer size.
  2. DMA controller takes control of the system bus and performs data transfer directly.

3. After completion, the DMA sends an **interrupt** to notify the CPU.

- **Advantages:**
  - **Very fast and efficient** for large data transfers.
  - Frees CPU for other tasks during data transfer.
- **Disadvantages:**
  - Requires special hardware (DMA controller).
  - May cause bus contention if multiple devices use DMA.
- **Example:**
  Disk-to-memory or memory-to-network data transfer.

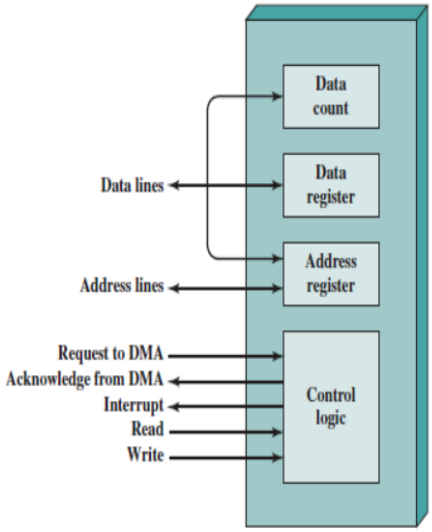
Comparison Summary

Feature	**Programmed I/O**	**Interrupt-Initiated I/O**	**Direct Memory Access (DMA)**
CPU Involvement	High (polling)	Medium (on interrupt)	Low (only at start/end)
Efficiency	Low	Moderate	High
Hardware Support	None	Interrupt controller	DMA controller
Suitable For	Simple, low-speed devices	Moderate-speed devices	High-speed, large transfers
Example	Keyboard input	Printer	Disk I/O

Basic Operation of DMA

When the processor wishes read or send a block of data, it issues a command to the DMA module by sending some information to DMA module. The information includes:

- read or write command, sending through read and write control lines.
- number of words to be read or written, communicated on the data lines and stored in the data count register.
- starting location in memory to read from or write to, communicated on data lines and stored in the address register.
- address of the I/O device involved, communicated on the data lines.



Typical DMA Block Diagram

After the information are sent, the processor continues with other work. The DMA module then transfers the entire block of data directly to or from memory without going through the processor. When the transfer is complete, the DMA module sends an interrupt signal to the processor to inform that it has finish using the system bus.

For configurations of DMA: https://inputoutput5822.weebly.com/direct-memory-access.html

II. IO Buffering

References:
https://www.geeksforgeeks.org/operating-systems/i-o-buffering-and-its-various-techniques/

I/O buffering is a technique in which a temporary memory area (buffer) is used to hold data while transferring between an I/O device and a process (CPU or memory).

It helps to match the speed difference between fast CPU/memory and slow I/O devices (like disks, printers, networks).

🔹 Why Buffering is Needed

- I/O devices are much slower than CPU.
- CPU should not sit idle waiting for I/O to finish.
- Buffering allows CPU and I/O devices to work in parallel.

Example:
When printing a document, data is first placed in a buffer so the CPU can continue working while the printer slowly prints the data.

How It Works

1. Application Process requests I/O (e.g., to read or write data).
2. Data is transferred to/from a buffer in main memory.
3. While one buffer is being filled or emptied by the I/O device, the process or CPU can continue using another buffer.

Types of Buffering

Type	Description	Diagram / Flow	Use Case
1. Single Buffering	Only one buffer in main memory. CPU fills or empties it while I/O device works.	CPU ↔ [Buffer] ↔ I/O Device	Simple systems; less efficient
2. Double Buffering (Ping-Pong Buffering)	Two buffers used alternately. While one is being filled, the other is being emptied.	[Buffer 1] 🔁 [Buffer 2]	Higher throughput; common in OS
3. Circular Buffering (Multiple Buffers)	More than two buffers arranged in a circular queue. Used when continuous data flow is required.	[B1 ➜ B2 ➜ B3 ➜ … ➜ B1]	Multimedia, real-time systems

---

Example – Double Buffering

Time	Buffer 1	Buffer 2	CPU State
t_1	CPU filling	I/O emptying	Both active

Time	Buffer 1	Buffer 2	CPU State
t_2	CPU emptying	I/O filling	Parallel work
t_3	CPU idle	Both filled/emptied	Synchronization phase

➡ This reduces CPU waiting time significantly.

Advantages of I/O Buffering

- Increases system throughput.
- Minimizes CPU idle time.
- Allows overlapping of computation and I/O.
- Smooths out variations in data flow rates.

Disadvantages

- Consumes extra memory space (for buffers).
- Adds management overhead for the OS.
- May cause latency if buffer sizes are not optimal.

III Hard Disk Architecture

References:

https://www.geeksforgeeks.org/computer-organization-architecture/hard-disk-drive-hdd-secondary-memory/

<Note: Check thru architecture, numericals and various terminologies >

IV. Disk Cache

A disk cache is a reserved portion of main memory (RAM) or a small amount of high-speed memory on the disk itself that temporarily stores frequently accessed data from the disk.

Its main purpose is to speed up data access by avoiding repeated slow reads/writes from the physical disk.

Where It Resides

There are two main types of disk caches depending on where they are located:

Type	Location	Managed by	Description
System (Software) Disk Cache	In main memory (RAM)	OS Kernel	Part of the system buffer cache or page cache that stores recent disk blocks.
Hardware Disk Cache	In the hard disk drive (HDD) or SSD	Disk controller (firmware)	Small built-in memory (e.g., 32MB–512MB) that caches data being read/written to improve drive performance.

How Disk Caching Works

1. When the CPU or process requests data from disk:
    - OS checks if data is already in disk cache (RAM).
    - If yes → Cache Hit → Data returned instantly from cache (fast).
    - If no → Cache Miss → Data fetched from disk, stored in cache for future use.
2. For writes, the OS may use write-back caching or write-through caching:
    - Write-Through: Data written to both cache and disk immediately (safer, slower).
    - Write-Back: Data written to cache first, then later to disk (faster, but risk of loss on power failure).

Disk Cache vs. Buffer Cache

Aspect	Disk Cache	Buffer Cache
Function	Caches disk data for faster access	Temporarily holds data during I/O transfers
Scope	File-level or block-level caching	Device I/O-level buffering
Location	In main memory (RAM) or drive memory	In OS memory (RAM)
Managed by	OS or Disk controller	OS kernel

In modern systems, both are often integrated under the OS page cache mechanism.

Benefits of Disk Caching

- Reduces disk access time.
- Improves overall system performance.
- Decreases wear and tear on physical drives.
- Enables smooth multitasking and faster file operations.