## Modes of Transfer

The primary modes of data transfer in computer architecture between the CPU and I/O devices are:

1. Programmed I/O
2. Interrupt-initiated I/O
3. Direct Memory Access (DMA).

## Programmed I/O

Programmed I/O operations are the result of I/O instructions written in the computer program.

Each data item transfer is initiated by an instruction in the program. Usually, the transfer is to and from a CPU register and peripheral.

Other instructions are needed to transfer the data to and from CPU and memory.

Transferring data under program control requires constant monitoring of the peripheral by the CPU.

Once a data transfer is initiated, the CPU is required to monitor the interface to see when a transfer can again be made.

It is up to the programmed instructions executed in the CPU to keep close tabs on everything that is taking place in the interface unit and the I/O device.

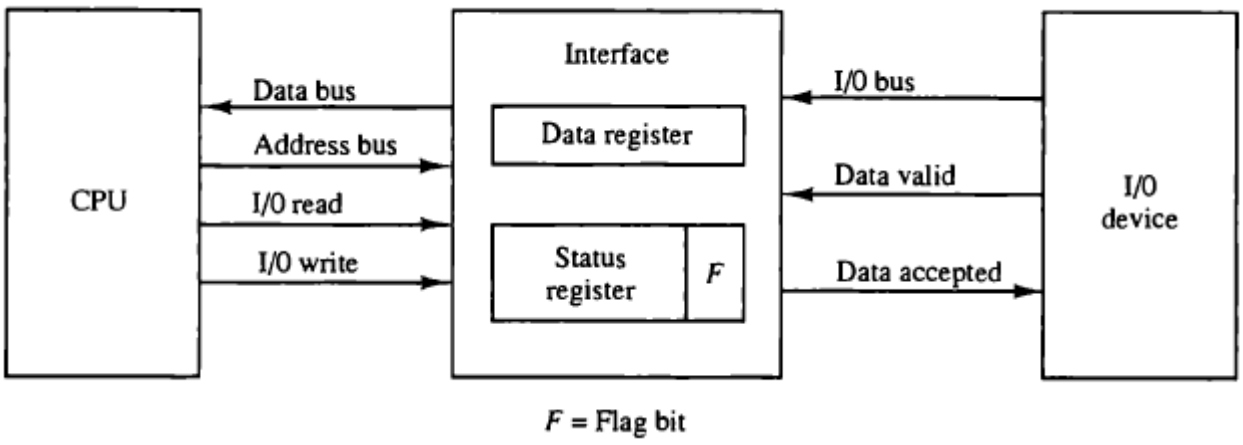


Figure: Data Transfer from I/O device to CPU

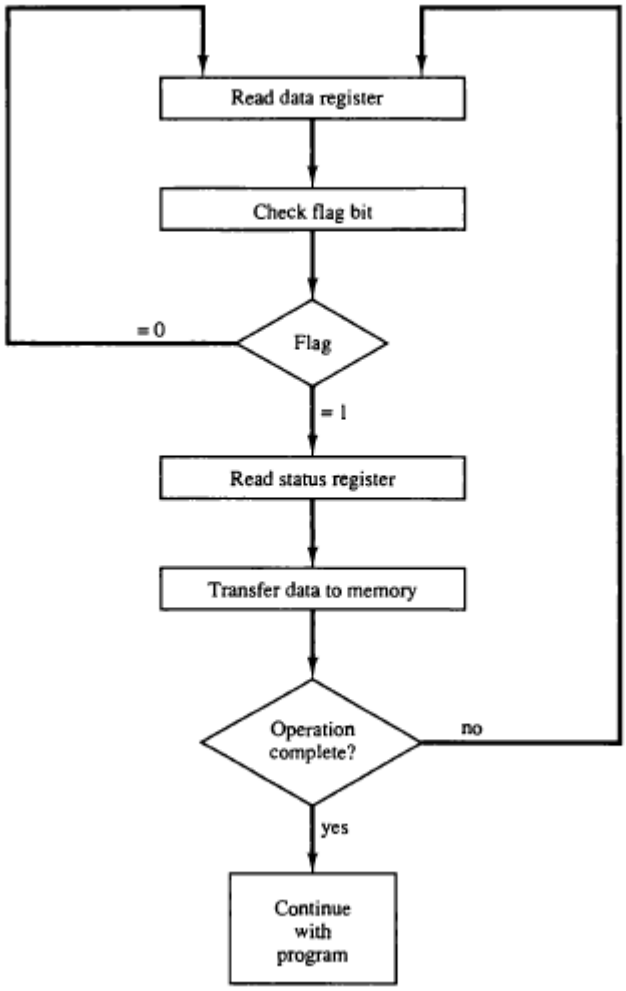



Figure: Flow-chart for CPU Program to input data

1. **Examples:**
    - Reading data from a keyboard or a mouse where the CPU continuously polls the device for keypresses or mouse movements.
    - Writing data to a printer where the CPU initiates the print operation, checks the printer status, and transfers data in small chunks.
2. **Drawbacks:**
    - **Inefficiency:** Programmed I/O can be inefficient, especially for high-speed devices or large data transfers, as it keeps the CPU busy and may lead to a waste of processing time.

- **Limited Concurrency:** The CPU is dedicated to managing the I/O operation, limiting its ability to perform other tasks concurrently.

## Interrupt-initiated I/O

Interrupt-initiated I/O is an input/output method where an I/O device signals the CPU by sending an interrupt request when it needs to transfer data, rather than the CPU constantly polling it. The CPU suspends its current task, executes an interrupt service routine (ISR) to handle the data transfer, and then resumes the original program. This approach is more efficient than programmed I/O because the CPU can perform other tasks while waiting for the device, leading to better utilization of its time and resources.

How Interrupt-Initiated I/O Works

1. **Device Ready Signal:**

   An I/O device, such as a keyboard or disk drive, generates an interrupt signal when it's ready to send or receive data.

2. **CPU Halts and Saves State:**

   Upon receiving the interrupt signal, the CPU momentarily stops executing the current program. It then saves the program's current state, including the program counter (the address of the next instruction), to a memory stack.

3. **Interrupt Service Routine (ISR):**

   The CPU branches to a special program called an Interrupt Service Routine (ISR) to handle the I/O operation.

4. **Data Transfer:**

   The ISR performs the necessary data transfer, either receiving data from an input device or sending it to an output device.

5. **Return to Original Program:**

Once the I/O operation is complete, the ISR restores the CPU's saved state, and the computer resumes execution of the program from where it left off.

Advantages

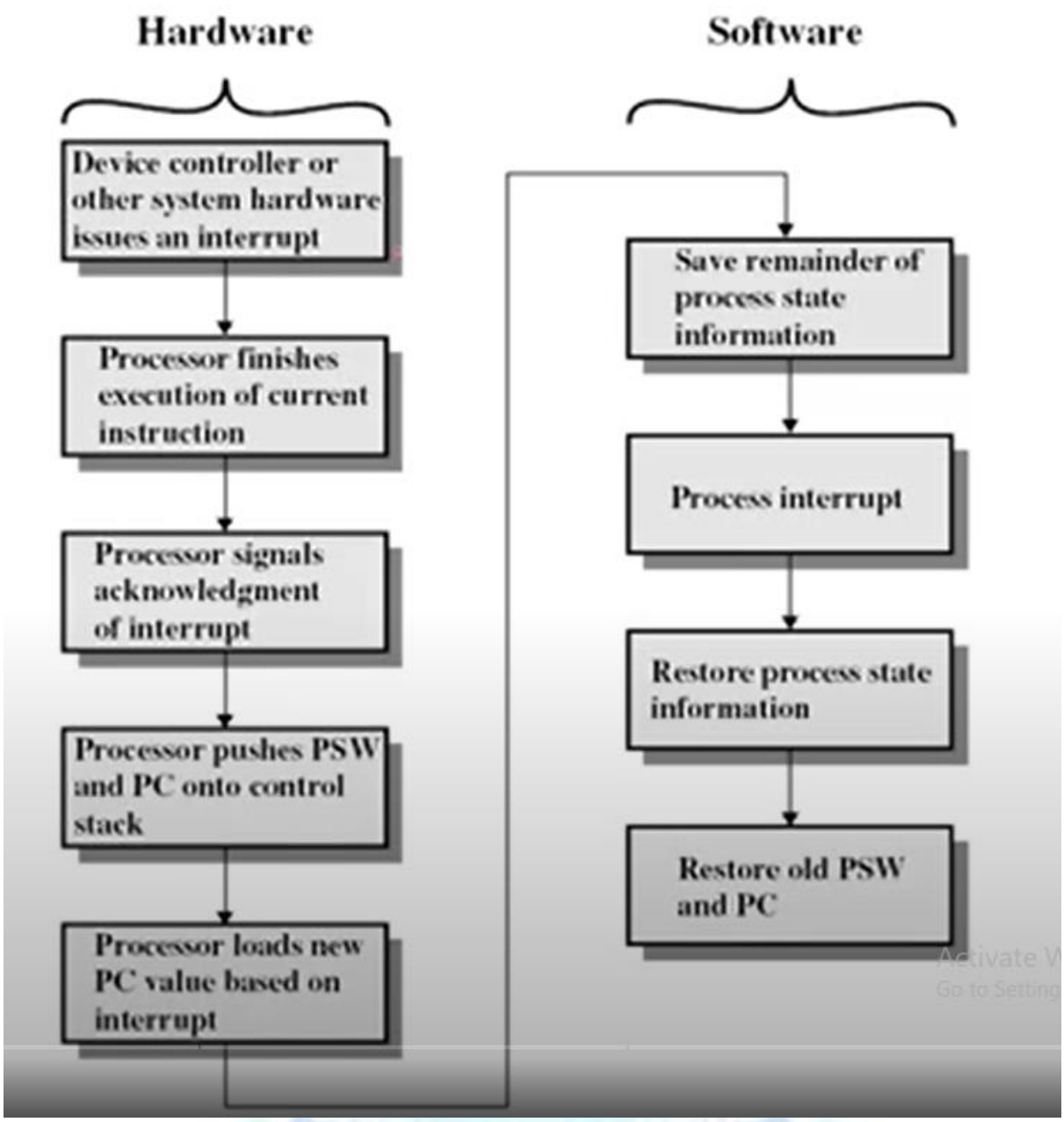
- **Efficiency:** The CPU doesn't waste time polling devices, allowing it to handle other tasks, which increases efficiency.
- **Responsiveness:** The system responds quickly to device requests because the device initiates the communication.
- **Resource Utilization:** The CPU's capacity and time are used more effectively.

Disadvantages

- **Complexity:**

  This method requires more complex hardware and software for managing interrupts.
- **CPU Involvement:**

  The CPU is still involved in every I/O transfer, unlike more advanced methods like Direct Memory Access (DMA).




**Priority Interrupt**

- Determines which **interrupt is to be served first** when **two or more requests** are made simultaneously
- Also determines which interrupts are permitted to interrupt the computer while another is being serviced
- Higher priority interrupts can make requests while servicing a lower priority interrupt

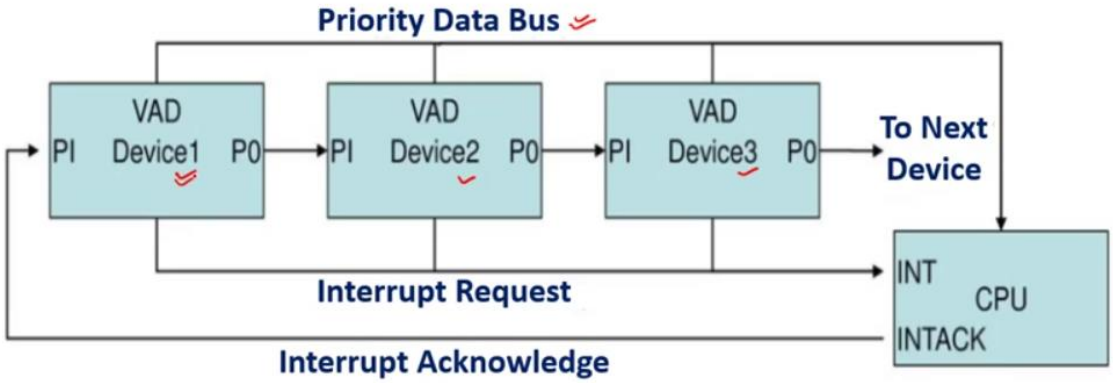
## Implementing Priority Interrupt by Software (Polling)

- **Priority** is **established** by the **order of polling the devices** (interrupt sources), that is identify the highest-priority source by software means
- One common branch address is used for all interrupts
- **Program polls** the interrupt sources in sequence
- The highest-priority source is tested first
- Flexible since it is established by software
- Very slow & Low cost since it needs a very little hardware

## Implementing Priority Interrupt by Hardware

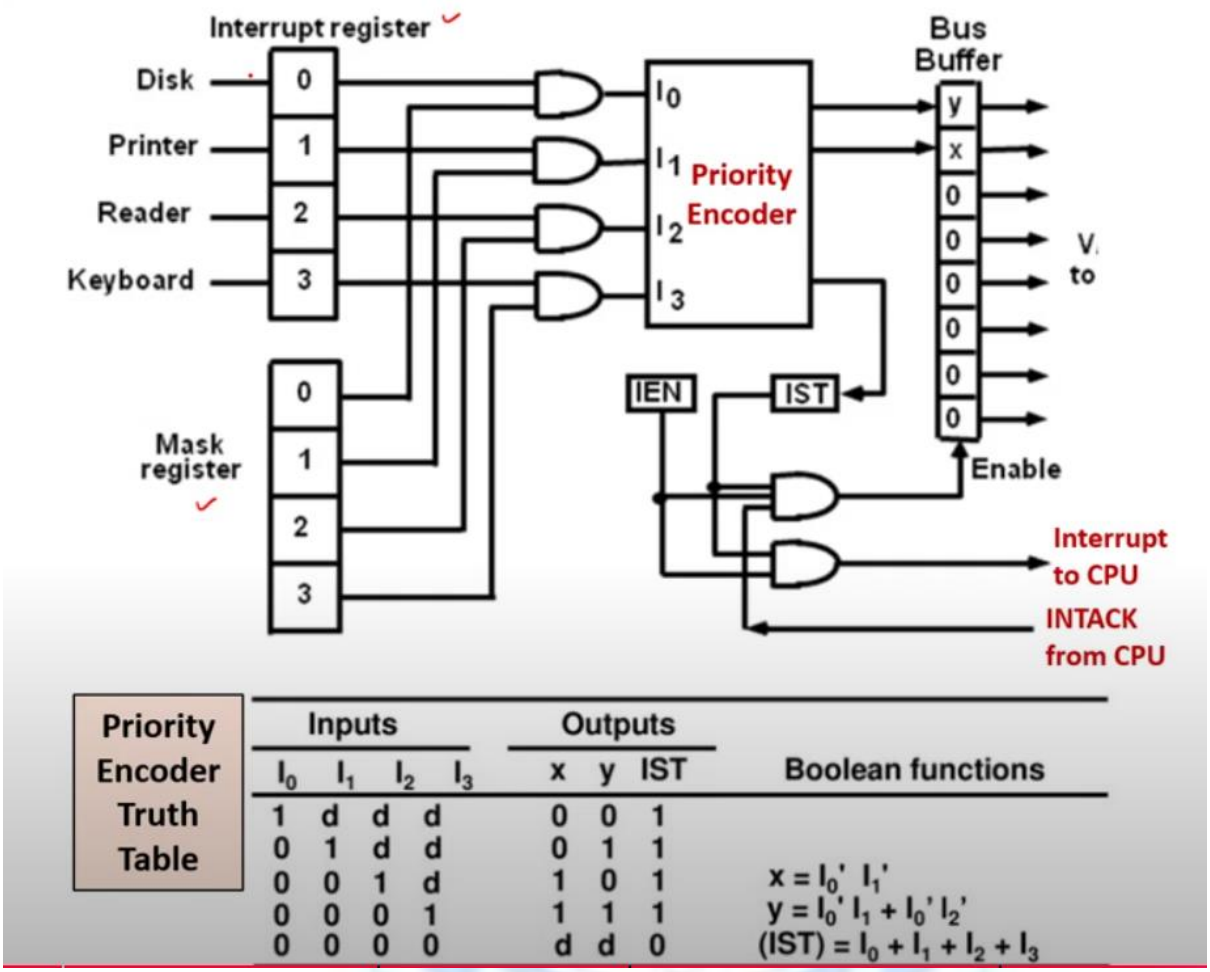
- **Require a priority interrupt manager** which accepts all the interrupt requests to determine the highest priority request
- Fast since identification of the highest priority interrupt request is identified by the hardware
- Fast since each interrupt source has its own interrupt vector to access directly to its own service routine

VAD: Vector Address




Daisy-Chain Priority Interrupt

## Parallel Priority Interrupt




**Priority Encoder Truth Table**

| Inputs | | | | Outputs | | | Boolean functions |
|---|---|---|---|---|---|---|---|
| $I_0$ | $I_1$ | $I_2$ | $I_3$ | x | y | IST | |
| 1 | d | d | d | 0 | 0 | 1 | |
| 0 | 1 | d | d | 0 | 1 | 1 | |
| 0 | 0 | 1 | d | 1 | 0 | 1 | $x = I_0' I_1'$ |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | $y = I_0' I_1 + I_0' I_2'$ |
| 0 | 0 | 0 | 0 | d | d | 0 | $(IST) = I_0 + I_1 + I_2 + I_3$ |

## Direct Memory Access

Direct Memory Access (DMA) is a technique used in computers and other electronic devices to allow peripherals (like hard drives, network cards, and sound cards) to communicate directly with the main memory (RAM) without involving the CPU. This process speeds up data transfer and frees up the CPU to perform other tasks, improving overall system performance.

- The peripheral device sends a request to the DMA controller to initiate a data transfer.

- The DMA controller takes control of the system's memory bus and accesses memory directly, either reading data from it or writing data to it.
- After the transfer is complete, the DMA controller signals the CPU that the task is finished, and the CPU can continue with other tasks.