Input/Output Systems and Peripheral Devices

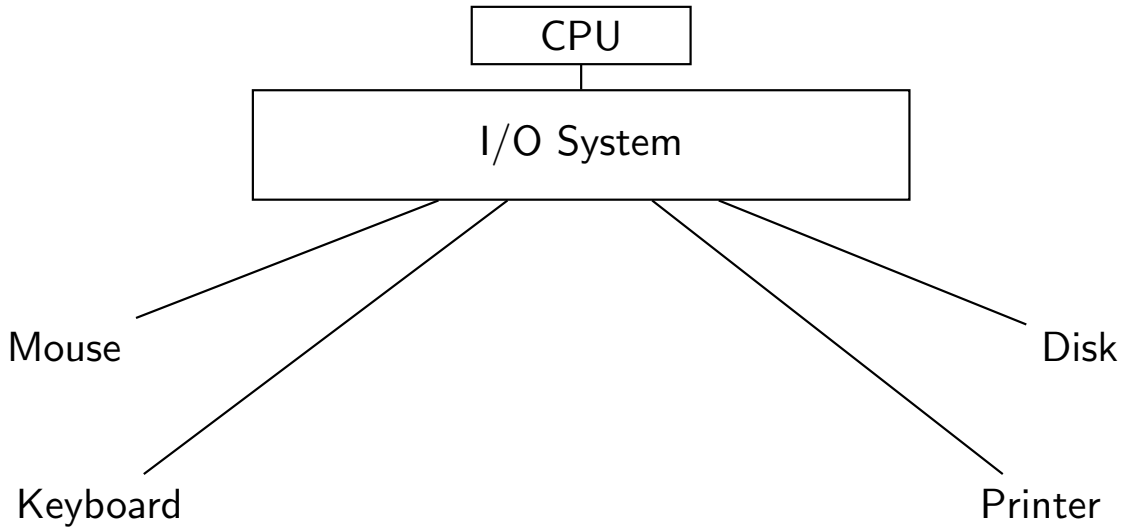
Srinivasan Ramachandran

April 7, 2025

Peripheral Devices

Categories of Peripheral Devices

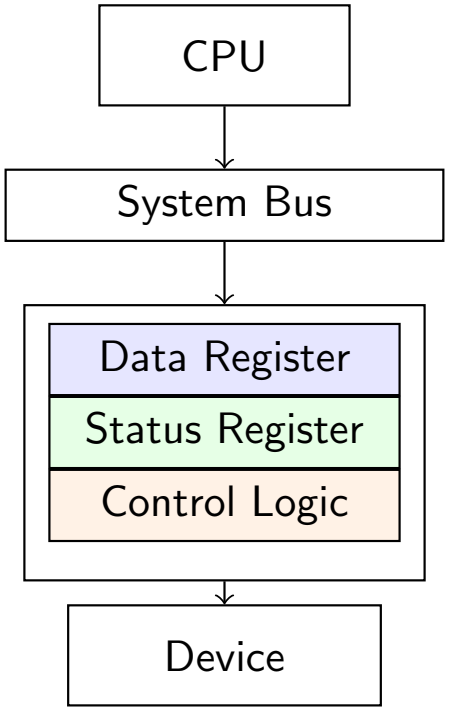
- Input Devices: Keyboard, Mouse, Scanner
- Output Devices: Monitor, Printer, Speakers
- Storage Devices: HDD, SSD, Flash Drives
- Communication Devices: NIC, Modem



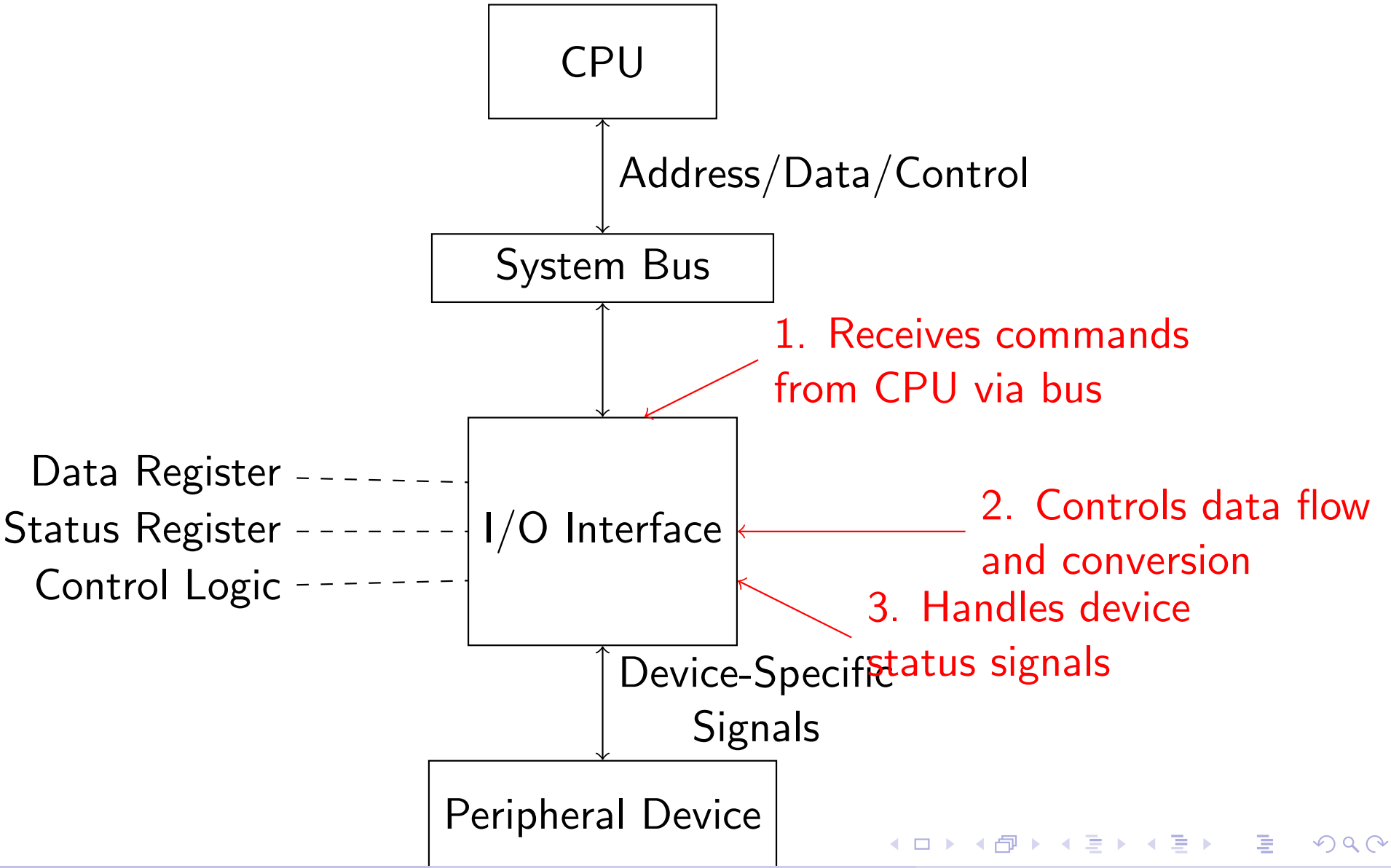
I/O Interface

Purpose of I/O Interface

- Resolve differences between CPU and peripherals
- Data format conversion (serial to parallel)
- Voltage level matching
- Protocol handling



I/O Interface Explained



Practice Problem 1: I/O Interface

Problem

An I/O interface has the following status register format:

- Bit 0: Device Ready (1 = ready)
- Bit 1: Error Flag (1 = error)
- Bit 2: Data Direction (1 = input)
- Bits 3-7: Reserved

Interpret the status register value `0x05` (00000101 in binary).

Practice Problem 1: I/O Interface

Problem

An I/O interface has the following status register format:

- Bit 0: Device Ready (1 = ready)
- Bit 1: Error Flag (1 = error)
- Bit 2: Data Direction (1 = input)
- Bits 3-7: Reserved

Interpret the status register value `0x05` (00000101 in binary).

Solution

- Binary: 00000101
- Bit 0 (Ready): 1 $\rightarrow$ Device is ready
- Bit 1 (Error): 0 $\rightarrow$ No error
- Bit 2 (Direction): 1 $\rightarrow$ Input operation
- The device is ready for input with no errors.

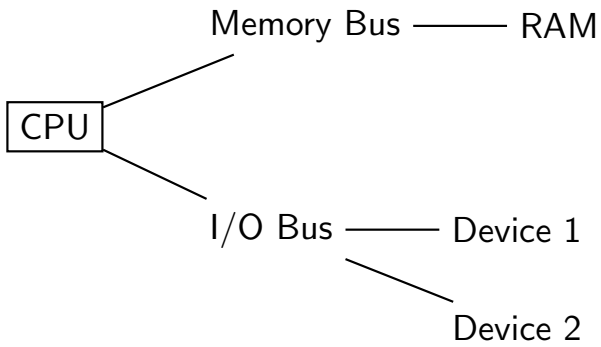
I/O Bus vs Memory Bus

I/O Bus Characteristics

- Connects CPU to peripherals
- Slower speed
- Supports multiple protocols
- Examples: USB, PCI, SATA

Memory Bus Characteristics

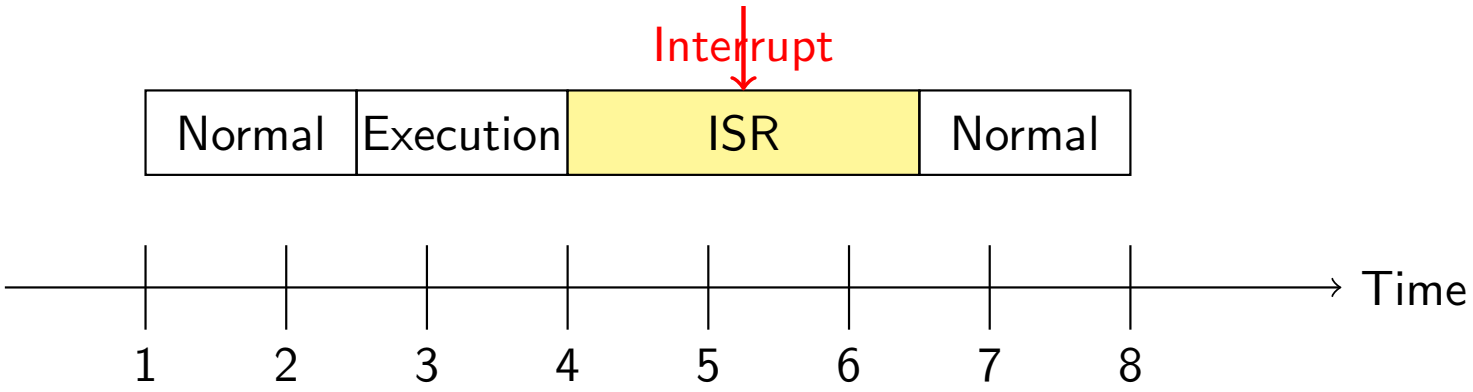
- Connects CPU to memory
- High speed
- Optimized for bulk transfers
- Examples: DDR, FSB



Interrupts and Types

Interrupt Handling Sequence

1. Device raises interrupt request
2. CPU finishes current instruction
3. CPU saves program state
4. CPU jumps to ISR (Interrupt Service Routine)
5. ISR executes and returns
6. CPU restores state and continues



Practice Problem 2: Interrupt Priorities

Problem

A system has three interrupt sources with the following characteristics:

Device	**Priority**	**ISR Time**	**Frequency**
Disk	High	50 μs	100/sec
Network	Medium	30μs	200/sec
Keyboard	Low	10μs	500/sec

Calculate the percentage of CPU time spent handling interrupts.

Practice Problem 2: Interrupt Priorities

Problem

A system has three interrupt sources with the following characteristics:

Device	**Priority**	**ISR Time**	**Frequency**
Disk	High	50 μs	100/sec
Network	Medium	30μs	200/sec
Keyboard	Low	10μs	500/sec

Calculate the percentage of CPU time spent handling interrupts.

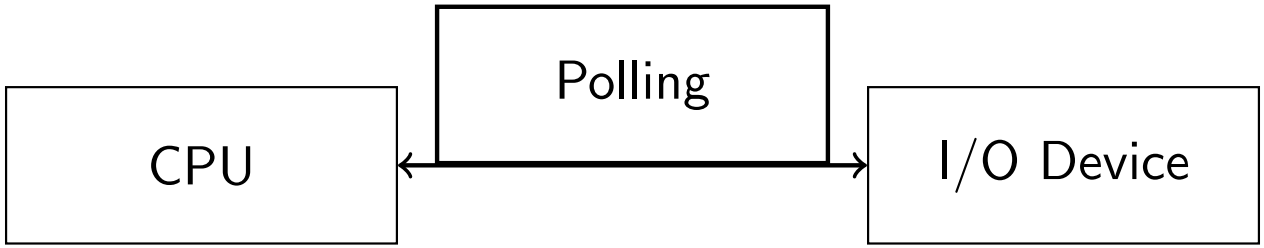
Solution

- Disk: 100 $\times$ \$50 μs = 5,000 μs
- Network: 200 $\times$ 30 μs = 6,000 μs
- Keyboard: 500 $\times$ 10 μs = 5,000 μs
- Total per second: 16,000 μs (1.6 % of CPU time)

Data Transfer Method: Programmed I/O

Programmed I/O

- CPU actively checks device status in a loop (polling)
- Simple to implement but inefficient
- High CPU involvement even when device is not ready
- *Example:* Polling a keyboard input in simple embedded systems

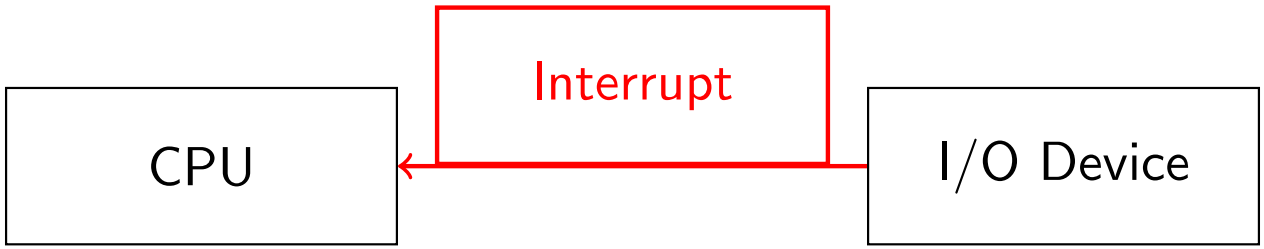


CPU repeatedly polls the device, consuming processing time regardless of I/O readiness.

Data Transfer Method: Interrupt-Driven I/O

Interrupt-Driven I/O

- Device notifies CPU via an interrupt signal when ready
- CPU pauses current task to handle I/O via an ISR (Interrupt Service Routine)
- Reduces CPU idle time compared to polling
- *Example:* Mouse movement or USB plug-in triggers interrupt

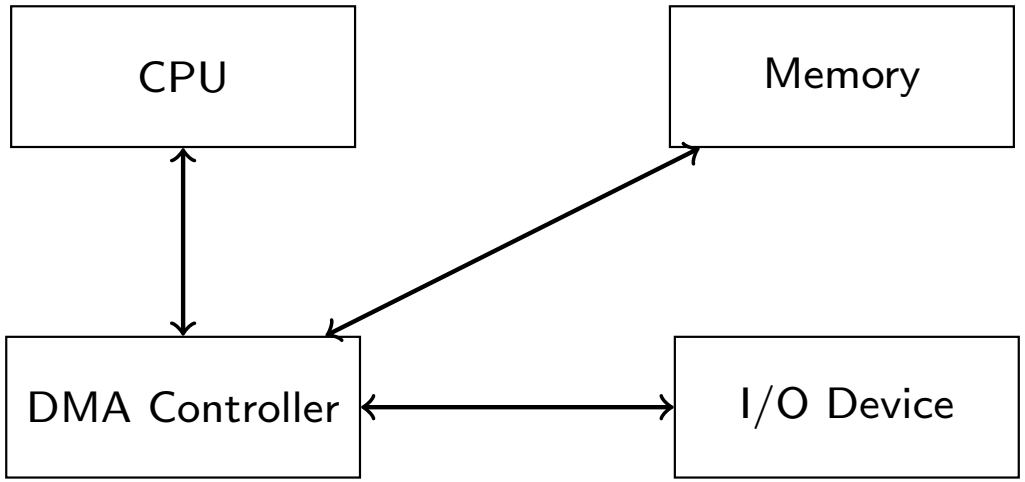


CPU only responds when the device signals readiness, improving efficiency.

Data Transfer Method: Direct Memory Access (DMA)

Direct Memory Access (DMA)

- DMA controller handles data transfer between memory and I/O
- CPU is involved only during initialization and completion
- Highly efficient for large/burst transfers (e.g., audio, video, sensors)
- *Example:* Transferring sensor data to memory in real-time systems

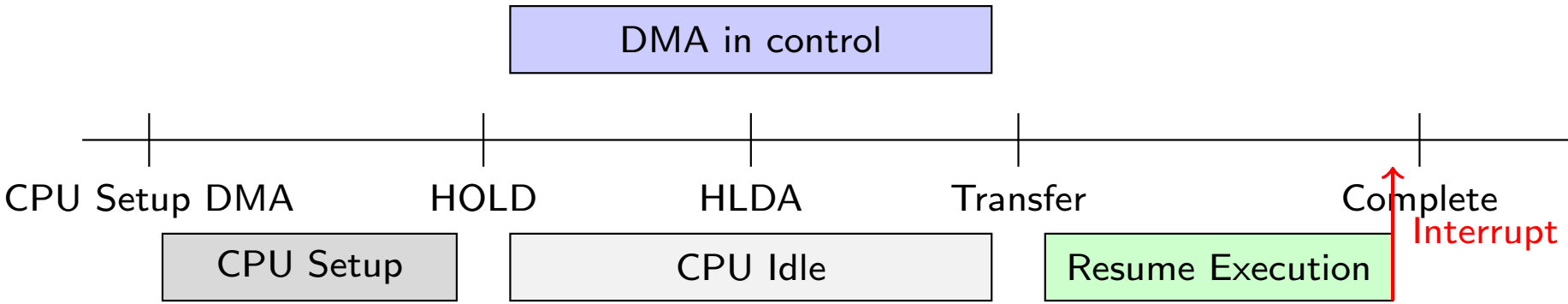


CPU delegates the transfer to DMA, which communicates directly between memory and device.

DMA Operation: Step-by-Step

1. **CPU initializes DMA Controller:**
   - Source and destination addresses
   - Number of bytes to transfer
   - Control mode (read/write, burst/single, etc.)
2. **DMA requests control of system bus (HOLD)**
3. **CPU acknowledges bus request (HLDA)**
4. **DMA performs direct memory transfer**
5. **DMA releases the bus**
6. **DMA sends an interrupt to CPU upon completion**

DMA Operation: Signal Timeline



This timeline shows how DMA takes control, performs transfer, and notifies CPU.
HLDA = Hold and Acknowledge

Why Use DMA?

- **Improves Performance:** Frees CPU from data transfer overhead.
- **Efficient for High-Speed I/O:** Ideal for audio, video, sensor streams.
- **Reduces Latency:** Allows parallel CPU and device operation.
- **Scalable:** Works with burst transfers and large data blocks.

DMA is critical for real-time and high-throughput embedded systems.

Practice Problem 3: DMA vs Programmed I/O

Problem

Compare the efficiency of transferring 1MB of data using:

- Programmed I/O: 1000 CPU cycles per byte
- DMA: 1000 cycles setup, 10 cycles per 512-byte block

Calculate the total cycles required for each method with a 1GHz CPU.

Practice Problem 3: DMA vs Programmed I/O

Problem

Compare the efficiency of transferring 1MB of data using:

- Programmed I/O: 1000 CPU cycles per byte
- DMA: 1000 cycles setup, 10 cycles per 512-byte block

Calculate the total cycles required for each method with a 1GHz CPU.

Solution

- Programmed I/O: 1,048,576 $\times$ 1000 = 1,048,576,000 cycles (1.05s)
- DMA: 1000 + (1,048,576/512 $\times$ 10) = 1000 + 20,480 = 21,480 cycles (21.48 μs)
- DMA is $\tilde{5}$0,000 times more efficient!

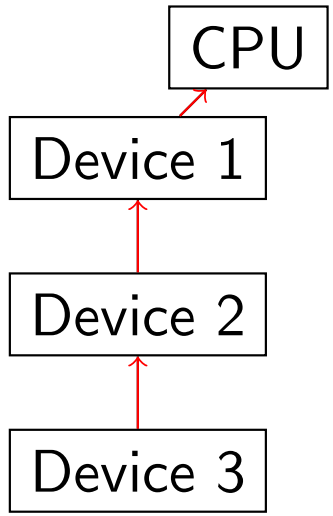
Priority Interrupt Systems

Daisy Chain

- Hardware priority
- Devices connected in series
- Closest to CPU has highest priority

Polling

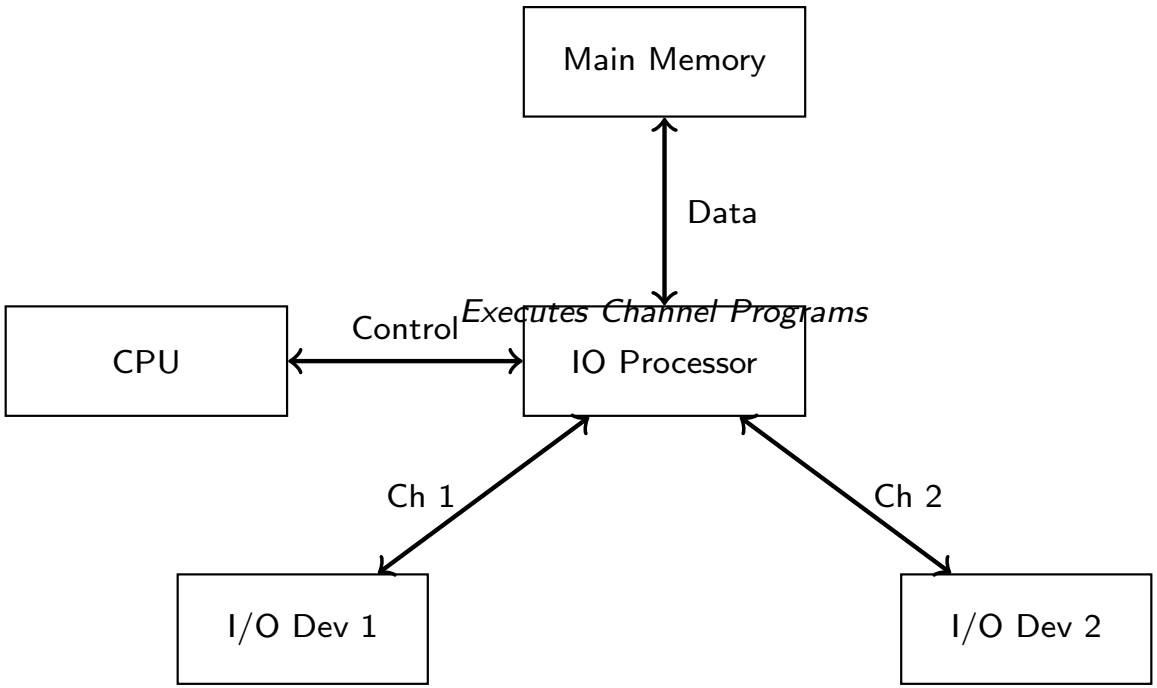
- Software priority
- ISR checks devices in order
- Flexible but slower



Input-Output Processor (IOP)

Why Use an IOP?

- Offloads routine and complex I/O processing from the CPU
- Executes **channel programs** independently
- Enables parallel I/O and computation
- Improves system efficiency and scalability



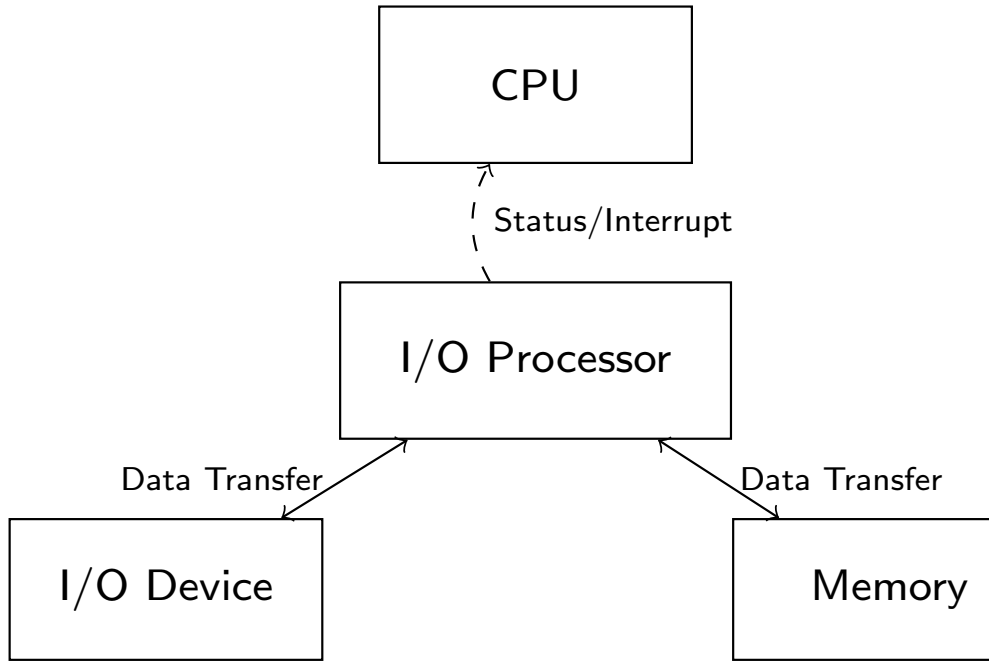
Channel Programs

What is a Channel Program?

A **channel program** is a sequence of I/O instructions executed by an I/O Processor (IOP) or channel subsystem. It manages data transfer between memory and I/O devices *independently of the CPU*.

Key Features

- Offloads I/O from CPU
- Executes multiple I/O steps
- Supports conditional logic
- Used in mainframes and DMA-capable systems



Practice Problem 4: IOP Performance

Problem

A system with IOP can process I/O operations in parallel with CPU execution. For a workload with:

- 60ms computation
- 40ms I/O (can be fully overlapped)

Compare execution time with and without IOP.

Practice Problem 4: IOP Performance

Problem

A system with IOP can process I/O operations in parallel with CPU execution. For a workload with:

- 60ms computation
- 40ms I/O (can be fully overlapped)

Compare execution time with and without IOP.

Solution

- Without IOP: $60 + 40 = 100$ms (sequential)
- With IOP: $\max(60,40) = 60$ms (parallel)
- 40% reduction in execution time

Key Takeaways

- I/O interfaces resolve device-CPU mismatches
- Interrupts improve efficiency over polling
- DMA minimizes CPU involvement in transfers
- Priority schemes manage multiple devices
- IOPs provide parallel I/O processing

Final Exam Question

Describe what happens when a keyboard key is pressed in a system using interrupt-driven I/O, from the electrical signal to the application receiving the character. Include all hardware and software components involved.