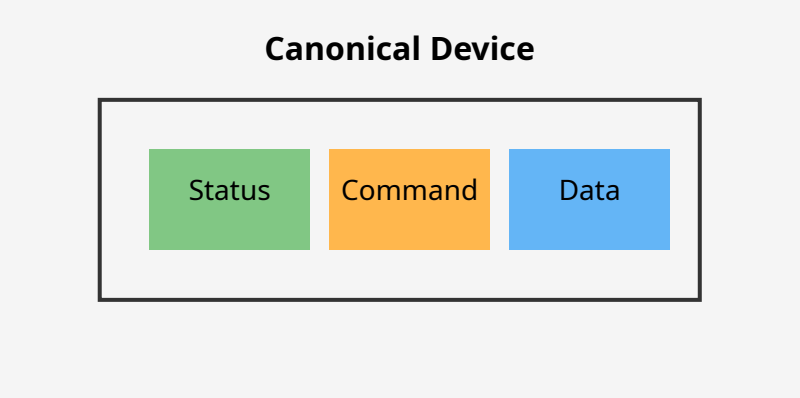
Chapter 36: I/O Devices

Overview

An I/O device, whether it's a hard drive, a keyboard, or a network card, has a standardized structure. It consists of two main parts: the hardware interface and the internal structure.

The **hardware interface** is what the rest of the system interacts with. It typically includes a set of registers that the CPU can read from and write to:

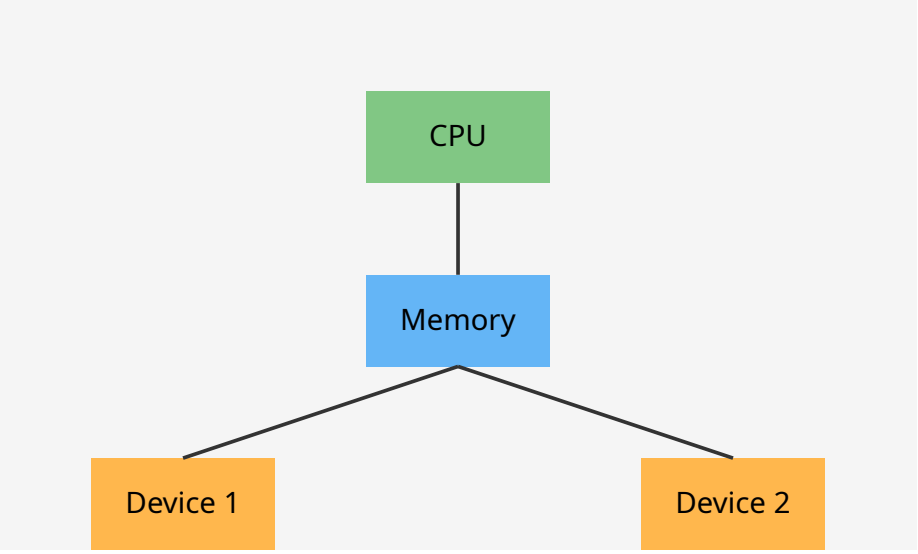
- **Status Register:** Indicates the current state of the device (e.g., busy, ready).
- **Command Register:** Used to tell the device what to do (e.g., read, write).
- **Data Register:** Used to pass data to or from the device.



The **internal structure** of the device is specific to its function. For example, a hard drive has a complex mechanical system for reading and writing data to platters, while a keyboard has a simpler mechanism for detecting key presses.

System Architecture

Devices are connected to the system via **buses**, which are electrical pathways that allow data to be transferred between different components. A typical system architecture looks like this:



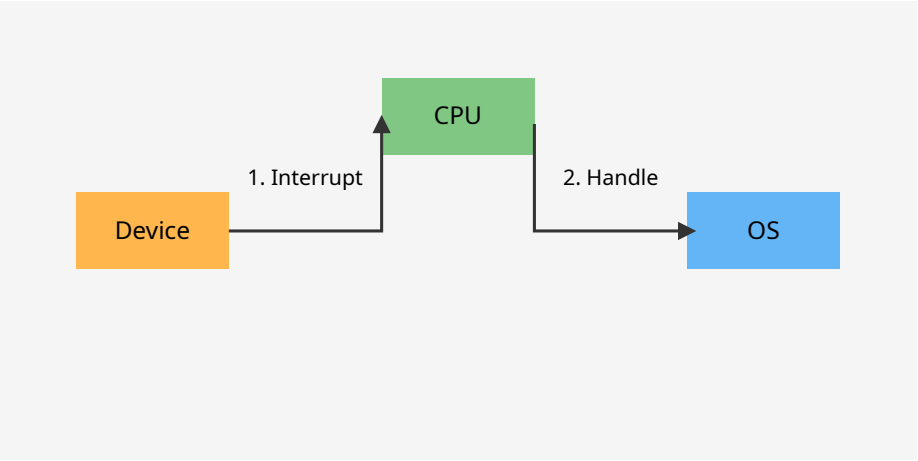
The CPU and memory are connected by a high-speed bus, while I/O devices are connected to a more general-purpose I/O bus, such as PCI. A **bus bridge** connects the different buses, allowing the CPU to communicate with the devices.

Communicating with Devices

There are three main ways the CPU can communicate with a device:

1. **Polling (Programmed I/O)**
   - With **polling**, the CPU repeatedly checks the device's status register to see if it's ready for the next command. This is simple to implement but can be very inefficient, as the CPU wastes a lot of time waiting for the device.

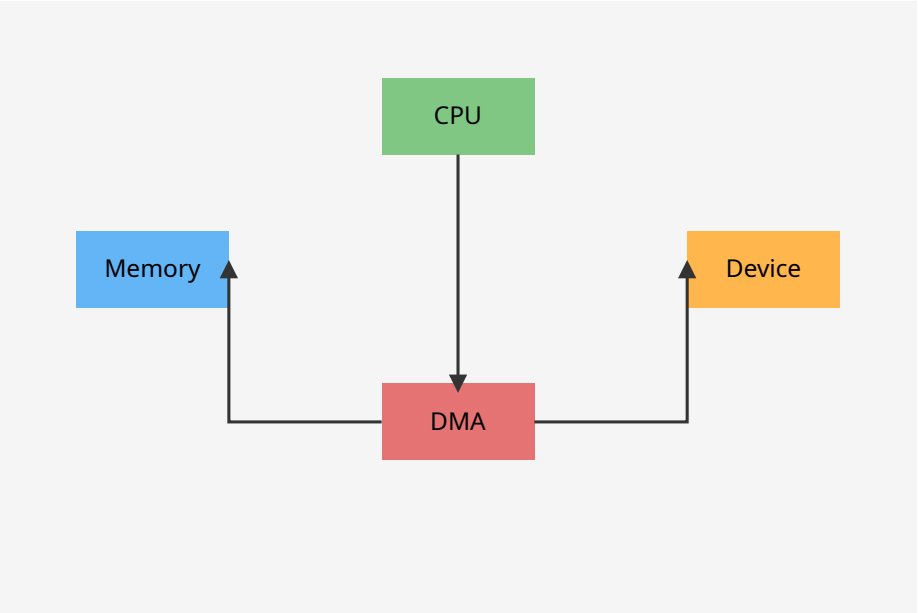
2. **Interrupts**
   - **Interrupts** provide a more efficient way for the device to notify the CPU that it's ready. When a device completes an operation, it sends an interrupt signal to the CPU. The CPU then suspends its current work and executes an **interrupt handler** to service the device.



- This is much more efficient than polling, as the CPU can perform other tasks while the device is busy.

3. Direct Memory Access (DMA)

- For devices that transfer large amounts of data, such as hard drives, even interrupts can be inefficient, as the CPU is involved in transferring every byte of data. **Direct Memory Access (DMA)** solves this problem by allowing the device to transfer data directly to and from memory, without involving the CPU.



The CPU initiates the transfer by telling the DMA controller what to do, and the DMA controller takes care of the rest. Once the transfer is complete, the DMA controller sends an interrupt to the CPU.

Device Drivers

- To hide the complexities of interacting with different devices, the operating system uses **device drivers**. A device driver is a piece of software that understands the specifics of a particular device and presents a standardized interface to the rest of the OS.

This abstraction is crucial for portability and maintainability. It allows the OS to support a wide variety of devices without having to be rewritten for each one.