Computer Organization & Architecture

Cybersecurity Department

Course Code: ***CBS219***

Lecture 7: Instruction Set Architecture (ISA) and I/O Organization

Halal Abdulrahman Ahmed

Lecture Outline

- I/O subsystem & need for I/O interfaces
- I/O controller: registers & functions
- System bus: address, data, control
- Bus read/write transactions (steps)
- I/O transfer methods: Polling, Interrupts, DMA
- Interrupt handling: ISR, latency, priority
- DMA: purpose & operation

Lecture Outcomes

By the end of this lecture, students will be able to:

- **Explain** the role of the I/O subsystem and why I/O controllers are essential.
- **Describe** the structure and purpose of the system bus (address, data, control).
- **Illustrate** step-by-step how memory read/write bus transactions occur.
- **Compare** polling, interrupt-based I/O, and DMA, and select the appropriate method for scenarios.
- **Define** interrupts, ISR, IRQ, and differentiate hardware vs software interrupts.
- **Explain** interrupt handling steps including context saving, latency, and priorities.
- **Describe** DMA operation and justify when DMA is preferred over interrupts or polling.

Introduction to Input-Output Interface

Input/Output (I/O) refers to all operations that transfer data between the computer and external devices.

- **Input devices** send data to the computer (keyboard, mouse, scanner).
- **Output devices** receive data from the computer (monitor, printer).
- Some devices do both (disk, USB storage, network card).
- I/O is not “extra”—it is a core part of computer architecture.

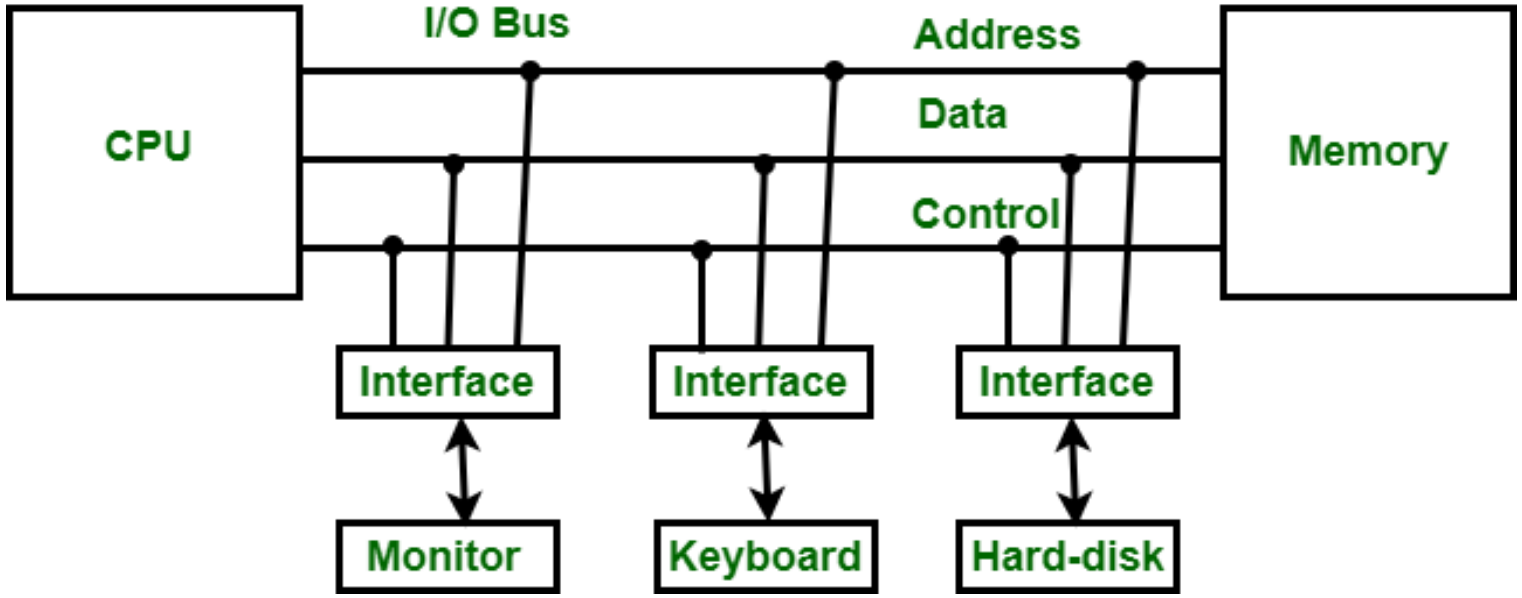
Why the CPU Cannot Communicate Directly with Devices

CPU and devices are incompatible in several ways:

- **Speed mismatch:** CPU is fast; many devices are slow → CPU cannot wait continuously
- **Different data formats:** serial vs parallel, device-specific encoding
- **Different control rules:** each device needs different command signals
- **Asynchronous behavior:** device events happen unpredictably (key press, network packet)
- So we insert hardware that “adapts” device behavior to the CPU/memory world.

I/O interface (controller/module)

An **I/O interface (controller/module)** is hardware that sits between the bus and the device. The CPU interacts with the controller using standard bus operations rather than talking to the device's internal electronics. The controller makes device communication predictable and manageable.



Inside an I/O Controller: Registers

Most controllers expose a small set of registers (seen by CPU):

- **Data Register:** holds incoming/outgoing data
- **Status Register:** flags like READY/BUSY/ERROR
- **Control Register:** commands like START/RESET/ENABLE-INTERRUPT
- These registers are the “language” between CPU and device.

Key Controller Functions

Controllers perform essential functions:

- **Timing & control:** generate proper device timing signals
- **Synchronization/handshaking:** coordinate slow/fast sides safely
- **Buffering:** temporary storage that prevents data loss
- **Error detection/reporting:** informs OS/CPU about failures
- **Data conversion (when needed):** serial↔parallel, ADC/DAC
- Important idea: buffering happens **inside the controller**, not “inside the bus”.

How CPU “Accesses” a Device

The CPU accesses **controller registers**, not the physical device directly. Two widely used approaches exist:

- **Memory-mapped I/O:** controller registers appear at specific memory addresses
- **Port-mapped I/O:** controller registers appear at I/O ports (separate addressing)
- For our course, focus on the idea: **an address selects a controller register**, then control signals determine read/write.

SYSTEM BUS FUNDAMENTALS

"Here comes the 64-bit local bus."

System Bus: Definition and Purpose

A **system bus** is a communication structure that carries:

- *Where* we access (addresses)
- *What* we transfer (data)
- *Which operation & timing* (control)
- It connects CPU, memory, and I/O controllers so the computer acts as one coordinated system.

Bus is Shared: Why Rules Are Needed

- A bus is often a **shared medium**. If two devices drive the data lines at the same time, signals conflict and data becomes unreliable. Therefore, systems enforce rules so only one device transmits at a time (arbitration). Shared bus requires coordination.

Address Bus

The **address bus** carries an address that identifies a destination:

- A memory location, or
- A controller register
- Usually it is **unidirectional** (CPU → others).
- Address width limits the maximum number of addressable locations.

Data Bus

The **data bus** carries the actual data values being transferred. It is typically **bidirectional**, because CPU both reads and writes data. A wider data bus transfers more bits per bus cycle, which can increase throughput.

Control Bus

The control bus carries signals that define:

- Type of operation (Memory Read, Memory Write, I/O Read, I/O Write)
- Timing/clock coordination
- Interrupt request/acknowledge
- Bus request/grant (when multiple masters exist)
- Without control signals, devices cannot know what to do, and transfers would be unsafe.

Bus Transaction Concept (One Transfer = Many Signals)

A “bus transfer” is not just sending data. It is a coordinated set of actions:

- Put address
- Assert control signals (read/write)
- Wait for device/memory response (ready)
- Transfer data

  This is why bus design directly affects system performance.

BUS READ AND BUS WRITE (STEP-BY-STEP)

When CPU reads from memory:

- CPU places address X on address bus
- CPU asserts **MEMORY READ** on control bus
- Memory decodes address and prepares the data
- Memory drives the data bus with the value
- CPU captures data into a register
- This is the hardware meaning of “load value from memory”.

Memory Write Transaction (Step-by-Step)

When CPU writes into memory:

- CPU places address X on address bus
- CPU places value V on data bus
- CPU asserts **MEMORY WRITE** on control bus
- Memory stores V at address X
- This is the hardware meaning of “store value in memory”.

I/O Read vs Memory Read

The bus procedure is similar, but the destination differs:

- Memory read: address selects a memory location
- I/O read: address selects a controller register

  In both cases, CPU uses address + control + data, but the target component changes.

Checkpoint: Bus Roles in One Sentence

If you can explain a read/write using these three, you understand bus basics.

- Address bus answers: Where?
- Data bus answers: What value?
- Control bus answers: Which operation and when?

I/O TRANSFER STRATEGIES

There are three common strategies for moving I/O data:

- **Programmed I/O (Polling)**
- **Interrupt-driven I/O**
- **DMA (Direct Memory Access)**

  They mainly differ in how much CPU is involved, and how efficiently time is used.

Programmed I/O (Polling): How It Works

In polling, the CPU repeatedly checks the device status register:

- If not ready → loop and check again
- If ready → transfer one unit of data

  This is simple and works in small systems, but wastes CPU cycles because CPU spends time waiting.

Polling: When It Is Acceptable

Polling may be acceptable when:

- The system is simple (embedded/small scale)
- The device becomes ready quickly
- Predictability is more important than CPU efficiency

  But in general-purpose computers, polling is inefficient for slow devices.

What is an Interrupt?

- An interrupt is a **signal** from hardware or software indicating an event that needs immediate attention. It temporarily **stops the CPU's current execution** so a high-priority task can be serviced. After handling the event, the CPU **resumes from the exact point** it was interrupted.
- Interrupts allow the CPU to do useful work instead of waiting. A device notifies the CPU only when service is needed. This improves throughput and responsiveness, especially when many devices exist.

How an Interrupt Works

- Device or software raises an **interrupt request (IRQ)**.
- CPU finishes the current instruction.
- CPU saves the **Program Counter (PC)** and registers.
- CPU loads the address of the **Interrupt Service Routine (ISR)**.
- ISR handles the event and clears the request.
- CPU restores context and continues execution.

Terminologies

- IRQ/INTR = **hardware signal line**
- ISR = **software routine** executed to handle the interrupt

Why Interrupts Are Needed

- Devices operate at slower speeds than CPU
- CPU must react immediately to certain events
- Reduces wasted time compared to polling
- Enables multitasking and responsive systems

Interrupt Latency

Interrupt latency = time between interrupt generation and start of ISR execution

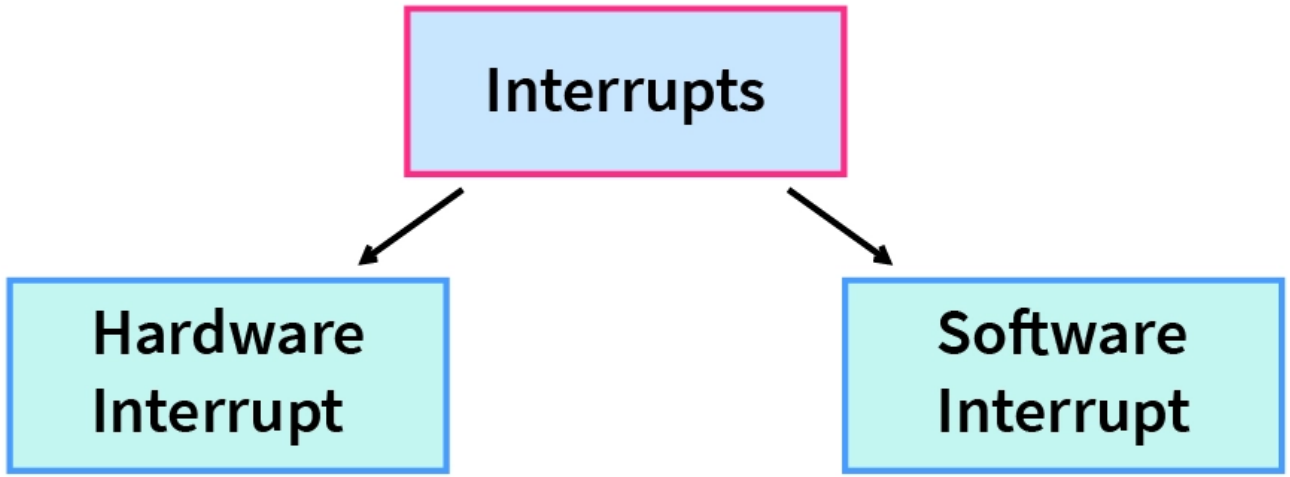
Increased by:

- Finishing current instruction
- Saving CPU registers
- Identifying interrupt source
- Priority checks
- Critical in **real-time systems**

Interrupt Types

There are two types of interrupts

1. Hardware interrupts
2. Software interrupts.



Hardware interrupts

- The interrupt signal generated from external devices and i/o devices are made interrupt to CPU when the instructions are ready. Hardware interrupts are classified into two types which are as follows:
- **Maskable Interrupt**: The hardware interrupts that can be delayed when a highest priority interrupt has occurred to the processor.
- **Non Maskable Interrupt**: The hardware that cannot be delayed and immediately be serviced by the processor.

For example: In a keyboard if we press a key to do some action this pressing of the keyboard generates a signal that is given to the processor to do action, such interrupts are called hardware interrupts.

Software interrupts

The interrupt signal generated from internal devices and software programs need to access any system call then software interrupts are present. Software interrupt is divided into two types. They are as follows:

1. **Normal Interrupts**: The interrupts that are caused by the software instructions are called software instructions.
2. **Exception:** Exception is nothing but an unplanned interruption while executing a program. For example: while executing a program if we got a value that is divided by zero is called an exception.

Sequences of Events Involved in Handling an IRQ(Interrupt Request)

- Devices raise an IRQ.
- The processor interrupts the program currently being executed.
- The device is informed that its request has been recognized and the device deactivates the request signal.
- The requested action is performed.
- An interrupt is enabled and the interrupted program is resumed.

Managing Multiple Devices

When more than one device raises an interrupt request signal, then additional information is needed to decide which device to be considered first. The following methods are used to decide which device to select: Polling, Vectored Interrupts, and Interrupt Nesting. These are explained below.

- **Polling:** In polling, the first device encountered with the IRQ bit set is the device that is to be serviced first. Appropriate ISR is called to service the same. It is easy to implement but a lot of time is wasted by interrogating the IRQ bit of all devices.

Managing Multiple Devices (cont.)

- **Vectored Interrupts:** In vectored interrupts, a device requesting an interrupt identifies itself directly by sending a special code to the processor over the bus. This enables the processor to identify the device that generated the interrupt. The special code can be the starting address of the ISR or where the ISR is located in memory and is called the interrupt vector.
- **Interrupt Nesting:** In this method, the I/O device is organized in a priority structure. Therefore, an interrupt request from a higher-priority device is recognized whereas a request from a lower-priority device is not. The processor accepts interrupts only from devices/processes having priority.

Edge-Triggered vs Level-Triggered

In digital circuits, two methods of triggering are possible, namely edge triggering and level triggering, which trigger the signal to switch from one state to the other. Both form part of digital electronics and help in increasing throughput and controlling the timing of operations in a given system.

- Level-triggered: interrupt request remains active until serviced
- Edge-triggered: interrupt occurs on a rising/falling edge (short event)

Direct Memory Access (DMA) Controller

In modern computer systems, transferring data between input/output (I/O) devices and memory can slow performance if the CPU manages every step. To overcome this, a Direct Memory Access (DMA) Controller is used. It enables I/O devices to transfer data directly to or from memory without heavy CPU involvement, improving speed and efficiency.

- It generates memory addresses and controls timing for efficient data movement.
- Once initialised by the CPU, it operates independently to complete the transfer.
- DMA = Data movement. Interrupt = Completion notification.

Types of Direct Memory Access (DMA)

There are four popular types of DMA.

- **Single-Ended DMA:** In this type, the DMA controller is connected only to one device (usually either the memory or the I/O device), and it directly controls data transfer.
- **Dual-Ended DMA:** The DMA controller is connected to both the source and the destination, typically memory and an I/O device.
- **Arbitrated-Ended DMA:** In systems with multiple DMA devices or masters, arbitration is needed to decide which device gets control of the bus. It is more advanced than Dual-Ended DMA.
- **Interleaved DMA:** Interleaved DMA are those DMA that read from one memory address and write from another memory address.

Working of DMA Controller

The DMA controller registers have three registers as follows.

- **Address register:** It contains the address to specify the desired location in memory.
- **Word count register:** It contains the number of words to be transferred.
- **Control register:** It specifies the transfer mode.

- All registers in the DMA appear to the CPU as I/O interface registers. Therefore, the CPU can both read and write into the DMA registers under program control via the data bus.
- The figure below shows the block diagram of the DMA controller. The unit communicates with the CPU through the data bus and control lines. Through the use of the address bus and allowing the DMA and RS register to select inputs, the register within the DMA is chosen by the CPU.
- RD and WR are two-way inputs. When BG (bus grant) input is 0, the CPU can communicate with DMA registers. When BG (bus grant) input is 1, the CPU has relinquished the buses and DMA can communicate directly with the memory.

How DMA Uses Interrupts

Once the CPU initializes the DMA controller, the **DMA performs the entire data transfer independently**.

- The CPU is **not involved** in reading or writing each byte.
- When the DMA finishes transferring the block of data:
- **DMA sends an interrupt to the CPU (DMA Completion Interrupt).**

This interrupt tells the CPU:

- The transfer is complete
- Memory contains the new data
- CPU can resume processing
- **DMA and Interrupts work together** to provide efficient I/O without CPU waiting.

References

- Fox, C. (2022). Computer architecture: From the Stone Age to the quantum age. No Starch Press.
- Stallings, W. (2022). Computer Organization and Architecture: Designing for Performance (11th ed.). Pearson.
- GeeksforGeeks. (n.d.). *External and internal interrupts*. https://www.geeksforgeeks.org/computer-organization-architecture/external-and-internal-interrupts/
- TutorialsPoint. (n.d.). *8086 Microprocessor — Interrupts*. https://www.tutorialspoint.com/microprocessor/microprocessor_8086_interrupts.htm


Any
Question