Computer Organization and Architecture

Input Output(I/O) Organization

Er. Sujan Karki (IOE Purwanchal Campus)

June 7, 2025

Introduction

In addition to the processor and a set of memory modules, the third key element of a computer system is a set of I/O modules. Each module interfaces to the system bus or central switch and controls one or more peripheral devices. An I/O module is not simply a set of mechanical connectors that wire a device into the system bus. Rather, the I/O module contains logic for performing a communication function between the peripheral and the bus.

Peripheral devices

- An external device connected to an I/O module is often referred to as a peripheral device or, simply, a peripheral.
- We can broadly classify external devices into three categories:
  - **Human readable:** Suitable for communicating with the computer user. e.g. video display terminal, printers etc.
  - **Machine readable:** Suitable for communicating with equipment. e.g. magnetic disk, magnetic tape, sensor, actuators used in robotics etc.
  - **Communication:** Suitable for communicating data with remote devices. e.g. modem, NIC (network interface Card) etc.

Block diagram of Peripheral Device

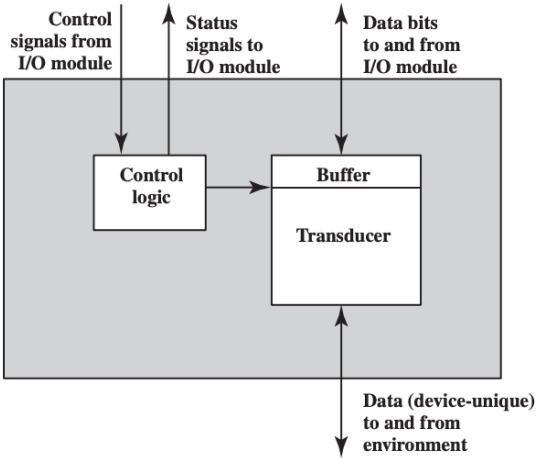


Figure 7.2 Block Diagram of an External Device

The interface to the I/O module is in the form of control, data, and status signals.

- **Control signals** determine the function that the device will perform, such as send data to the I/O module (INPUT or READ), accept data from the I/O module (OUTPUT or WRITE), report status, or perform some control function particular to the device (e.g., position a disk head).
- **Status signals** indicate the state of the device. Examples are READY/ NOT-READY to show whether the device is ready for data transfer.
- **Data** are in the form of a set of bits to be sent to or received from the I/O module.
- The **transducer** converts data from electrical to other forms of energy during output and from other forms to electrical during input.
- Typically, a **buffer** is associated with the transducer to temporarily hold data being transferred between the I/O module and the external environment.

- **Input Devices**
  - Keyboard
  - Mouse
  - Touch Screen
  - Light Pen
  - Scanner
  - Microphone
  - Webcam
  - Joystick
  - Barcode Reader
  - Optical Character Recognition (OCR)
  - Optical Mark Reader (OMR)
  - Digitizer
- **Output Devices**
  - Monitor (CRT, LCD, LED)
  - Printer (Inkjet, Laser, Dot Matrix)
  - Speakers
  - Headphones
  - Plotter
  - Projector

I/O Modules

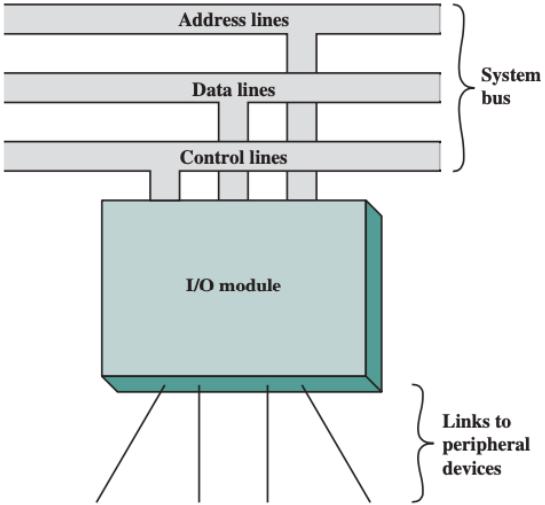


Figure 7.1 Generic Model of an I/O Module

I/O Module (Input/Output Module) is a hardware component in a computer system that connects the CPU and memory to peripheral devices like keyboards, printers, and disks. It acts as a communication bridge between the processor and external devices.
Peripherals are not directly connected to the system bus instead an I/O module is used which contains logic for performing a communication between the peripherals and the system bus.
The reasons due to which peripherals do not directly connected to the system bus are:

1. There are a wide variety of peripherals with various methods of operation. It would be impractical to incorporate the necessary logic within the processor to control a range of devices.
2. The data transfer rate of peripherals is often much slower than that of the memory or processor. Thus, it is impractical to use the high-speed system bus to communicate directly with a peripheral.
3. On the other hand, the data transfer rate of some peripherals is faster than that of the memory or processor. Again, the mismatch would lead to inefficiencies if not managed properly.(NVMe SSDs)
4. Peripherals often use different data formats and word lengths than the computer to which they are attached. Thus, an I/O module is required.

I/O Modules Structure

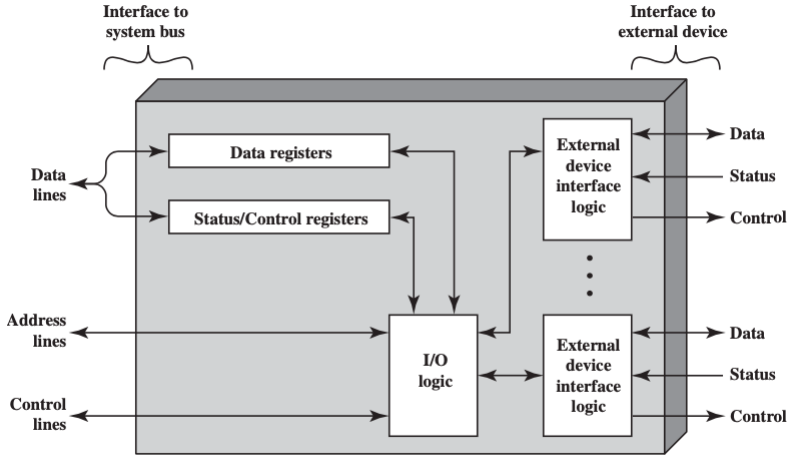


Figure 7.3 Block Diagram of an I/O Module

I/O Modules Functions

The detailed functions of I/O modules are;

1. Control & Timing
2. Processor communication
3. Device communication
4. Data buffering
5. Error detection

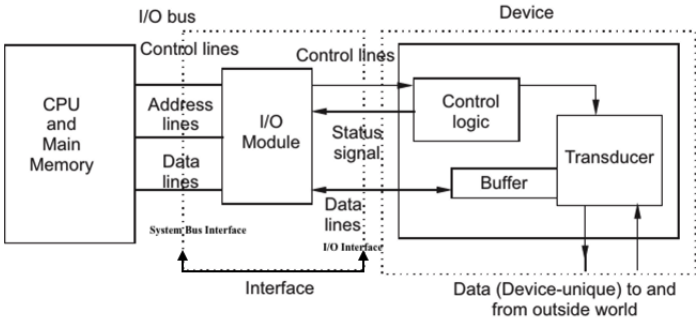
1. **Control & Timing**
   I/O module includes control and timing to coordinate the flow of traffic between internal resources(such as main memory and the system bus) and external devices. The control of the transfer of data from external devices to processor consists following steps:
   a. The processor interrogates the I/O module to check the status of the attached device.
   b. The I/O module returns the device status.
   c. If the device is operational and ready to transmit, the processor requests the transfer of data, by means of a command to the I/O module.
   d. The I/O module obtains a unit of data (e.g., 8 or 16 bits) from the external device.
   e. The data are transferred from the I/O module to the processor.
2. **Processor Communication**
   I/O module communicates with the processor which involves:
   1. **Command decoding:** I/O module accepts commands from the processor.
   2. **Data:** Data are exchanged between the processor and I/O module over the bus.
   3. **Status reporting:** Peripherals are too slow and it is important to know the status of I/ O module. Common status signals are BUSY and READY.
   4. **Address recognition:** I/O module must recognize one unique address for each peripheral it controls.

3. **Device Communication**
   It involves commands, status information and data.
4. **Data Buffering:** I/O module must be able to operate at both device and memory speeds.
   - When the CPU sends data to a device:
     - Data is sent quickly to the I/O module.
     - The I/O module stores (buffers) the data.
     - Then it sends it slowly to the device at the device's speed.
     - E.g. CPU to Printer
   - When the device sends data to memory:
     - The I/O module receives and buffers the slow input.
     - Then sends it in bursts to memory, avoiding slowdowns.

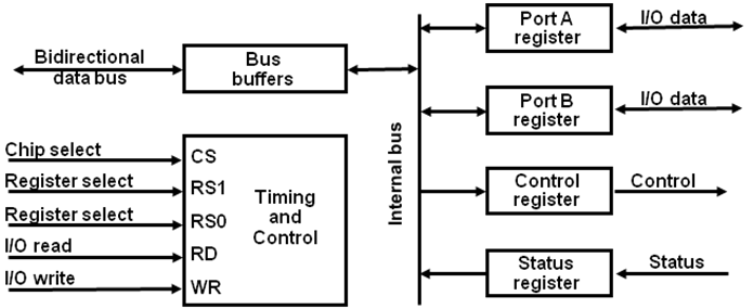
   Hence, buffering ensures smooth communication between fast and slow parts of the system.
5. **Error Detection:** I/O module is responsible for error detection such as mechanical and electrical malfunction reported by device e.g. paper jam, bad ink track & unintentional changes to the bit pattern and transmission error. Along with reporting errors to the processor.

Input-Output interface

An **I/O interface** refers to the set of protocols, standards, and connectors used to connect external devices to a computer system.
It includes physical connectors (such as USB interface, HDMI interface, Ethernet ports) and communication protocols that allow devices like keyboards, mice, monitors, printers, and networking equipment to communicate with the computer.



I/O Interface



CS	RS1	RS0	Register selected
0	x	x	None - data bus in high-impedence
1	0	0	Port A register
1	0	1	Port B register
1	1	0	Control register
1	1	1	Status register

Interface performs the following:

- Decodes the device address (device code)
- Decodes the commands (operation)
- Provides signals for the peripheral
- Synchronizes the data flow and supervises the transfer rate between peripheral and CPU or Memory

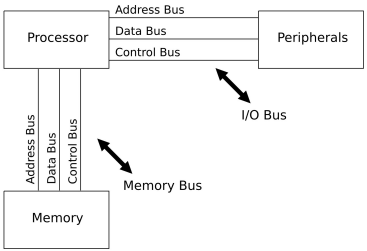
I/O commands that the interface receives:

- Control command: issued to activate the peripheral and to inform it what to do.
- Status command: used to test various status conditions in the interface and the peripheral.
- Output data: causes the interface to respond by transferring data from the bus into one of its registers.
- Input data: It is the opposite of the data output. In this case the interface receives an item of data from peripheral and places it in it's buffer register.

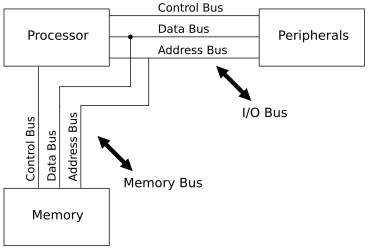
I/O Bus versus Memory Bus

As a CPU needs to communicate with the various memory and input-output devices (I/ O) as we know data between the processor and these devices flow with the help of the system bus. There are three ways in which system bus can be allotted to them :

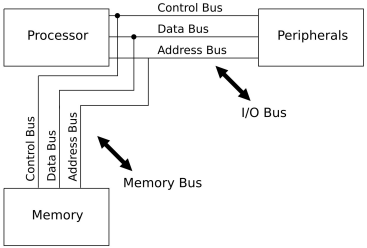
1. Separate sets of address, control, and data buses are used for I/O and memory. This method allows the I/O processor to handle input/output operations independently of the CPU.
2. I/O and memory share the same data and address buses, but use separate control lines to distinguish between I/O and memory operations (Isolated I/O Method)
3. I/O and memory share the same data, address, and control buses. I/O devices are assigned unique memory addresses, and standard memory access instructions are used to communicate with them (Memory Mapped I/O method)



Separate Buses



Common Bus with Separate Control Lines



Common Bus with Shared Control Lines

Modes of Transfer

Various modes exist for transferring data between the central computer and I/O devices. There are three primary modes through which data is transferred from peripheral devices to the CPU.

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

Programmed I/O

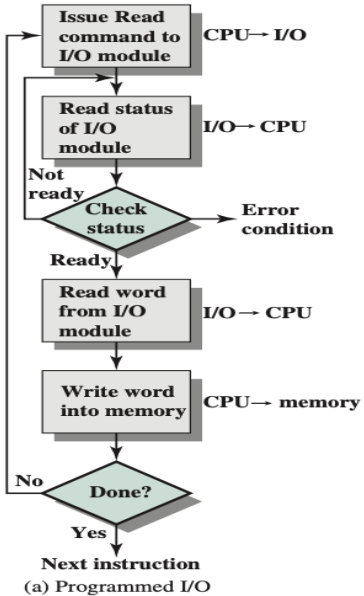
- The programmed I/O was the most simple type of I/O technique for the exchanges of data or any types of communication between the processor and the external devices.
- It is the result of instructions written in the program, i.e., each data transfer is initiated by an instruction in the program.
- The I/O device does not have direct access to memory; all executions happen via CPU operations.
- The CPU issues a command and then waits for the I/O operation to complete.
- Since the CPU is much faster than the I/O module, the main drawback of programmed I/O is that the CPU has to wait a long time for the I/O module to become ready for either data reception or transmission.
- While waiting, the CPU must repeatedly check the status of the I/O module. This process is known as polling.
- As a result, the overall performance of the system is significantly degraded.

Characteristics:

- Continuous CPU involvement in I/O operations
- CPU speed is effectively slowed down to the speed of the I/O device
- Simple to implement
- Requires the least hardware support

Applications of Programmed I/O:

- Useful in small, low-speed computers where speed requirement are not critical.
- Used in systems dedicated to continuously monitoring a device.for example, a sensor.



(a) Programmed I/O

Interrupt-driven I/O

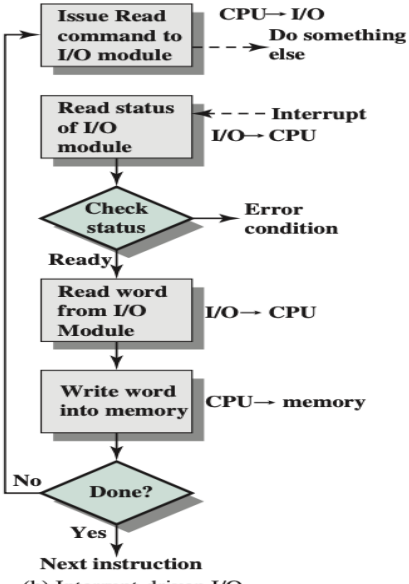
- Interrupt-driven I/O is an alternative method for handling input/output operations.
- It is a mechanism in which a peripheral device that needs to send or receive data signals the CPU by generating an interrupt.
- Polling, which involves repeatedly checking the status of an I/O device, consumes valuable CPU time. Interrupt-driven I/O avoids this overhead by initiating communication only when data needs to be transferred.
- In this method, the I/O interface — not the CPU — monitors the status of the I/O device.
- When the interface detects that the I/O device is ready for data transfer, it sends an Interrupt Request (IRQ) to the CPU.
- Upon receiving the interrupt, the CPU temporarily pauses its current task, executes the Interrupt Service Routine (ISR) to handle the I/O, and then resumes the interrupted task.
- With interrupt-driven I/O, the CPU can issue an I/O command and continue executing other tasks without waiting.
- The I/O module interrupts the CPU when it is ready to exchange data, requesting service. The interrupt can be initiated by either hardware or software.

Characteristics:

- More complex than programmed I/O, but much more efficient.
- Supports asynchronous I/O communication.
- The CPU is not constantly involved in monitoring I/O devices.
- Efficient CPU utilization — the CPU can perform other tasks while waiting.

Applications of Programmed I/O:

- Real-time systems, where timely response to input/output is crucial (e.g., industrial control, robotics).
- Multitasking environments, where the CPU needs to handle multiple tasks efficiently.



(b) Interrupt-driven I/O

Simple Interrupt Processing

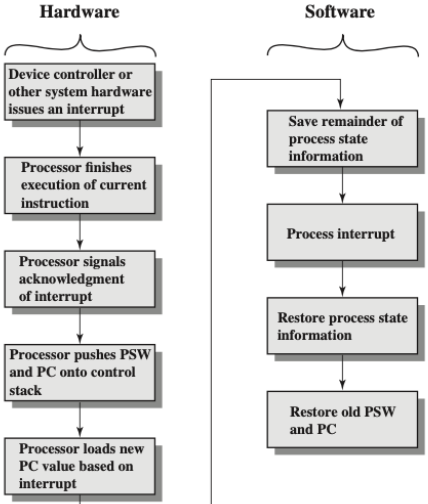


Figure 7.6 Simple Interrupt Processing

- Device triggers an interrupt – A hardware device signals the CPU for attention.
- CPU finishes current instruction – CPU completes the instruction it's working on.
- CPU acknowledges the interrupt – CPU sends a signal back confirming receipt of the interrupt.
- CPU saves current state (PSW & PC) – Pushes status word and program counter to the stack.
- CPU loads ISR address – Sets the PC to the starting address of the interrupt handler.
- After the CPU saves the basic state (PSW and PC), the software (Interrupt Service Routine, or ISR) saves any additional data that the program was using — so that it can safely return to the interrupted task later. Eg. CPU registers (like accumulator, index registers),General-purpose variables, Temporary data in memory, Flags not included in PSW etc.
- Process interrupt: The interrupt is handled (e.g., data is read or sent)
- Restore process state information: Saved CPU states are reloaded.
- Restore old PSW and PC: CPU resumes the original program from where it was interrupted.

Design Issues

There are 2 main problems for interrupt I/O, which are:

- Identifying the interrupting device:
  - With many I/O modules, how does the CPU know which device sent the interrupt?
- Prioritizing multiple simultaneous interrupts:
  - If more than one device interrupts at once, which one should be serviced first?

There are 4 main ways to counter these problems, which are:

- Multiple Interrupt Lines
- Software Poll
- Daisy Chain (Hardware Poll, Vectored)
- Bus Arbitration (Vectored)

Multiple Interrupt Lines

- As the name suggests, we provide multiple interrupt lines between the processor and the I/O modules.
- This allows multiple modules to be handled at the same time. However, it is not practical to assign many bus lines and processor pins to interrupt lines.
- One of the reasons is that there might be more than one I/O module attached to a single line.
- This defeats the purpose of this technique.
- The 3 techniques of the latter are usually used together with this technique to rectify its' problems.

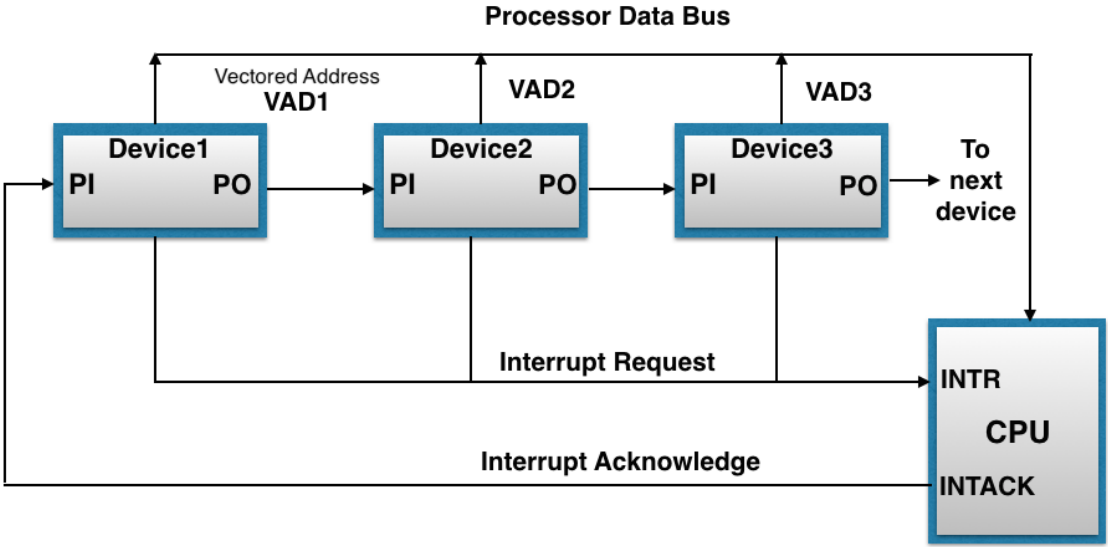
Priority Interrupt by Software (Polling)

- Software Polling is a technique where the CPU checks each I/O device in a fixed order (usually using a loop in software) to determine which one requested an interrupt.
- When multiple devices can cause interrupts, and there's no hardware priority mechanism, software is used to poll the devices one by one to:
  - Identify the source of the interrupt.
  - Determine priority based on the order in which devices are checked in the code.
- How It Works?
  - A device triggers an interrupt.
  - CPU acknowledges the interrupt and transfers control to a common Interrupt Service Routine (ISR).
  - Inside the ISR, the CPU runs polling code to:
    - Check each device's status register, in a fixed order.
    - Detect which device actually caused the interrupt.
  - The ISR then calls or jumps to the appropriate device-specific handler.
  - After servicing, the CPU resumes the interrupted program.

- Advantages:
  - Simple to implement.
  - No additional hardware required.
  - Priorities are easy to control and change in code so flexible.
- Disadvantages:
  - CPU time is wasted checking idle devices. So, very time consuming.
  - Not suitable for time-critical operations.
  - Response time can be slow if low-priority device is polled late.
- Use cases:
  - Embedded systems with a few I/O devices.
  - Systems where interrupts are too expensive or unavailable.
  - Low-speed peripheral management where efficiency is less critical.

Priority Interrupt by Hardware (Daisy Chain)

- Daisy Chaining is a hardware-based method for handling priority-based interrupt requests from multiple devices. Devices are connected in series (like a chain), where the interrupt acknowledgment signal (INTA) is passed from one device to the next.
- It is often used in systems with vectored interrupts, where each device provides its own interrupt vector (address of the interrupt service routine).
- So what is vector? A memory address that points to the Interrupt Service Routine (ISR) — the specific piece of code that handles the interrupt.
  - The interrupting device provides a vector (i.e., address) to guide the CPU directly to the appropriate ISR.
  - When an interrupt occurs, the CPU needs to know what code to run to handle it. Instead of the CPU figuring it out or using a fixed location, the device itself supplies the vector (address).This saves time and makes interrupt handling faster.


Processor Data Bus
Vectored Address
VAD1
VAD2
VAD3
Device1
PI
PO
Device2
PI
PO
Device3
PI
PO
To
next
device
Interrupt Request
INTR
CPU
Interrupt Acknowledge
INTACK

Components Involved:

- **Interrupt Request Line (INTR):** Shared line used by all devices to signal the CPU.
- **Interrupt Acknowledge Line (INTACK):** Passed sequentially from one device to another in the chain.

Operation Flow:

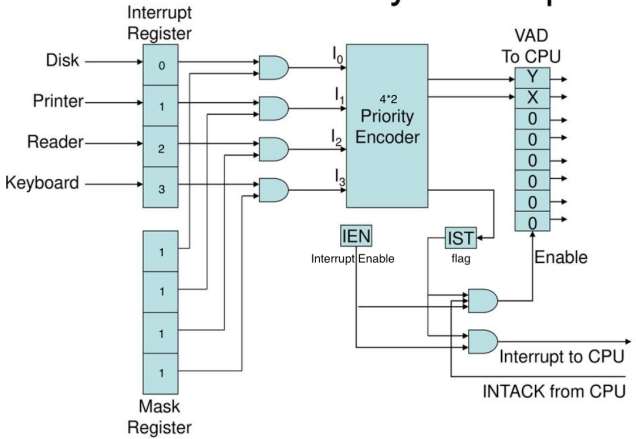
1. One or more devices raise an interrupt via the INTR line.
2. CPU detects the interrupt and sends an INTACK (Interrupt Acknowledge) signal.
3. The first device in the chain checks:
   - If it requested the interrupt:
     - Captures INTACK
     - Sends its interrupt vector to the CPU
     - Blocks INTACK from reaching other devices
   - If it did not request the interrupt:
     - Passes INTACK to the next device in the chain
4. CPU receives the interrupt vector and jumps to the corresponding ISR (Interrupt Service Routine).

- Advantages:
  - No Software Polling as devices handle priority in hardware.
  - Faster as devices provide the ISR address directly. (Vectored)
  - Saves CPU time as CPU doesn't check each device. It only send INTACK signal.
- Disadvantages:
  - Fixed Priority as closest device to CPU always has the highest priority. Not flexible.
  - Chain Dependency as failure in one device may block others from responding.
  - Propagation Delay as acknowledge signal passes sequentially—slower for lower-priority devices.
- Use cases:
  - Embedded systems where hardware-based priority is acceptable.
  - Microprocessor systems like 8085, 8086 with limited interrupt lines.

Parallel Priority

- Parallel Priority Interrupt is a hardware interrupt handling technique where multiple devices can simultaneously request an interrupt, and a priority encoder hardware circuit determines which interrupt has the highest priority in parallel (at the same time), rather than sequentially checking devices one after another.
- Devices (e.g., Disk, Printer, Reader, Keyboard) are connected to the interrupt system.
- Each device has a line connected to the Interrupt Register — 1 = request interrupt, 0 = idle.
- The Mask Register stores one bit per device to enable (1) or disable (0) interrupt requests from that device.
- The Interrupt Enable (IEN) bit determines whether any interrupt will be allowed at all.
- IST flag denotes that whether any device generate interrupt or not.. It is 1 for interrupt and 0 for no interrupt.

Parallel Priority Interrupt



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(DMA)

- DMA is a technique used in computer systems to facilitate data transfers between an I/O device and memory without involving the CPU extensively.
- Data transfer between I/O devices and memory is slow due to the speed mismatch in their speeds. This is avoided by removing the CPU from the path and letting the peripheral device manage the memory buses directly. This "data transfer between I/O device and memory without the involvement of CPU is called Direct Memory Access or DMA".

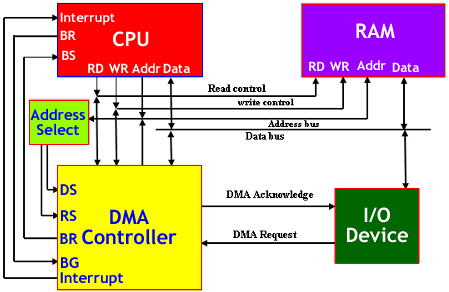


Fig: Showing DMA Mode of Data Transfer

There are two main types of DMA transfer methods based on how the DMA controller interacts with the system bus and CPU:

- Burst Mode DMA (Block Transfer Mode)
  - The DMA controller takes full control of the system bus and transfers the entire block of data in one go, without CPU intervention.
  - CPU is halted during the transfer.
  - Fastest method for transferring large blocks of data.
  - CPU cannot access the bus until the transfer is complete.
- Cycle Stealing DMA
  - DMA controller "steals" one bus cycle at a time from the CPU to transfer one word or byte of data.
  - CPU and DMA take turns using the bus.
  - CPU is slowed down but not completely halted.
  - Better for systems that need real-time CPU access.

Working

- **DMA Request from I/O Device:** The I/O device that needs to transfer data sends a DMA Request (DREQ) signal to the DMA controller.
- **DMA Requests Bus from CPU:**
  - Upon receiving the request, the DMA controller sends a HOLD (bus request) signal to the CPU.
  - CPU finishes its current instruction and: Sends back a Hold Acknowledge (HLDA) signal and disables its access to the system bus, allowing DMA controller to take control.
- **CPU Initializes the DMA Controller:** The CPU programs the DMA controller by sending:
  - Start address of memory block (where data should be read from or written to).
  - Word count (total number of bytes or words to transfer).
  - Control bits (to specify transfer mode: Read or Write).
  - Bus Grant $= 1$, which officially gives the DMA controller control over address, data, and control buses.

- **DMA Data Transfer Begins:** DMA controller now has control of the system bus. It sends a DMA Acknowledge (DACK) signal to the peripheral device.
  The device:
  - Puts data on the data bus (if writing to memory), or Reads data from the data bus (if reading from memory).

  The DMA controller:
  - Transfers one word at a time.
  - Increments address.
  - Decrements word count.
- **Completion of Transfer:** When the word count reaches 0, the DMA controller:
  - Stops the transfer.
  - Deactivates DACK signal to the peripheral.
- **Notify CPU :** DMA controller sends an Interrupt to inform the CPU that the transfer is complete.
  - Now CPU Disables the Bus Grant, taking back control of the bus.
  - Resumes normal instruction execution.

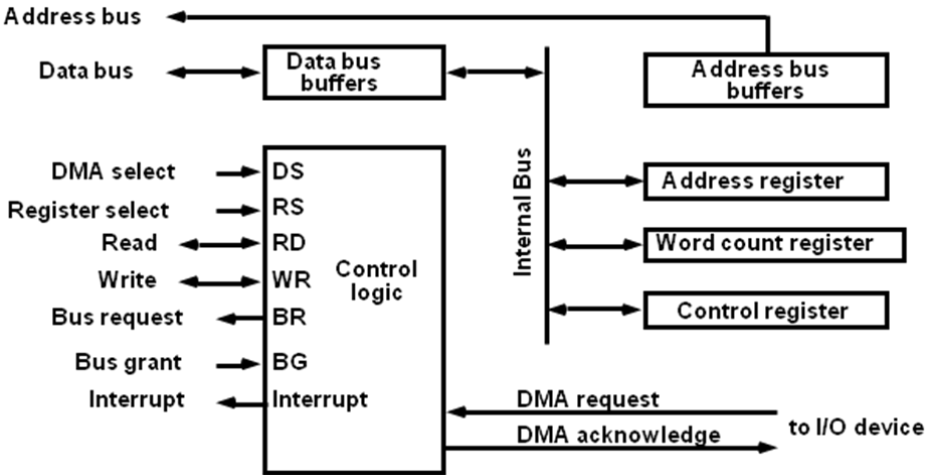


Figure: Block diagram of DMA Controller

I/O Processors

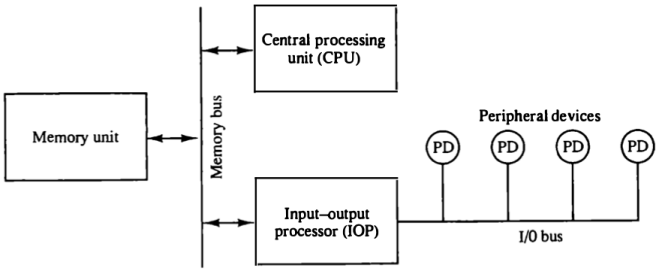
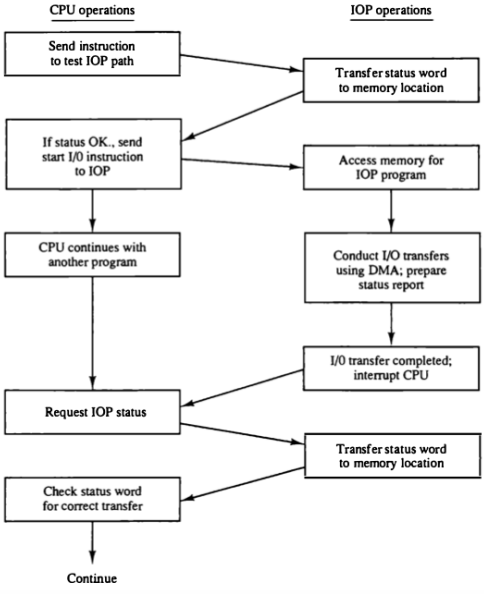


Figure 11-19 Block diagram of a computer with I/O processor.

- It is a specialized processor in a computer system that handles input/output (I/O) operations.
- The IOP fetches and executes I/O instructions to facilitates I/o transfer. If required, it can perform all the functions of CPU.
- Both CPU and IOP exists in the system however CPU is the master and IOP is slave.
- The CPU only initiates the I/O program after that IOP operates independent of the CPU.

Figure 11-20 CPU-IOP communication.



1. **CPU sends a test instruction to IOP**
   - To checks if the IOP is connected and working properly.
2. **IOP sends a status word to memory**
   - IOP responds by writing a status word (like a report) into a known memory location.
   - This tells the CPU whether IOP is ready or has any error.
3. **CPU reads the status word from memory**
   - If everything is OK, CPU sends a "start I/O" instruction to the IOP (like: "Start reading data from disk").
4. **CPU continues with its own program**
   - While the IOP handles I/O, the CPU keeps running its own instructions (no need to wait).
5. **IOP reads the program/instruction from memory**
   - IOP fetches the necessary I/O commands from memory: What device to talk to, Where to store or read data, How many bytes to transfer etc.

6. **IOP performs I/O transfer using DMA**
   - It uses Direct Memory Access (DMA) to send or receive data without disturbing the CPU. While doing this, it creates a status report.
7. **When finished, IOP interrupts the CPU**
   - IOP sends a signal (interrupt) to say: "Hey CPU, I'm done with the I/O task!"
8. **CPU asks IOP for final status**
   - CPU sends a request to get the final result of the I/O transfer.
9. **IOP writes final status word into memory again**
   - Tells whether the I/O was successful or had any error.
10. **CPU reads the status word and checks**
    - If transfer was successful $\rightarrow$ continue.
    - If error $\rightarrow$ handle it accordingly.

Data Communication Processor

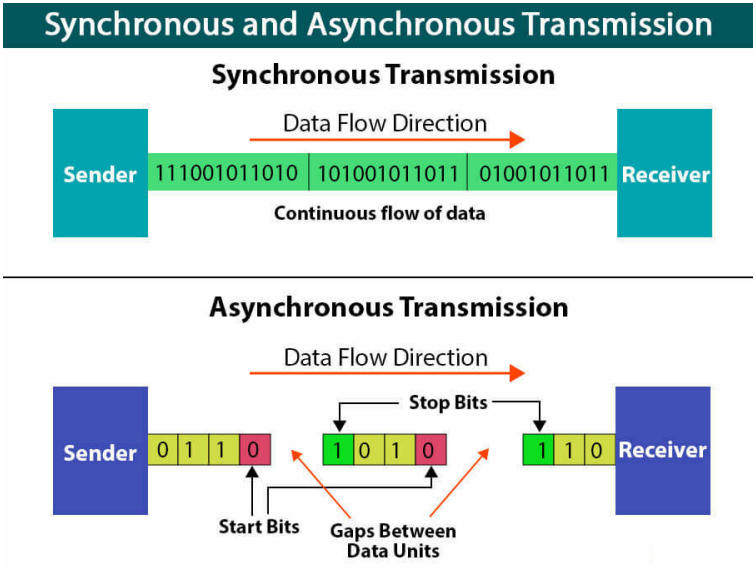
- A Data Communication Processor is a specialized hardware component or module designed to manage communication between a computer system and external devices or networks. Its main job is to handle the data transmission and reception processes, often taking over these tasks from the main CPU to improve efficiency.
- This processor ensures that data sent and received over networks is transmitted correctly and efficiently. It may also perform error detection and correction, data formatting, and protocol management.
- The data communication processor communicates with CPU and memory similar to any I/O processor.
- Unlike I/O processors communicating through a common bus, data communication processors communicate with each terminal through a single pair of wires.

1. Transmission

Data transmission between devices can be done in two main ways:

- **Synchronous Transmission :** Data is sent in a continuous stream, synchronized by a clock signal shared between the sender and receiver. Both devices agree on timing beforehand, so data arrives at fixed intervals.
  - Suitable for high-speed communication.
  - Requires synchronization mechanisms.
  - Example: Communication between computers in a LAN.
- **Asynchronous Transmission :** Data is sent one byte (or character) at a time with start and stop bits to indicate the beginning and end of each unit. No clock signal is shared; the receiver uses these bits to identify data boundaries.
  - Suitable for slower or intermittent communication.
  - No need for synchronization.
  - Example: Keyboard to computer communication

Synchronous and Asynchronous Transmission



2. Transmission Errors (Error Detection Methods)
   - **Parity Check :** Adds a single parity bit to data to make the total number of 1s either even (even parity) or odd (odd parity). Simple but not reliable for detecting multiple-bit errors.
   - **Checksum :** Summation of all data bytes transmitted; the sender calculates it and sends it along with data. The receiver recalculates the sum and compares it with the received checksum. If different, an error is detected.
   - **Cyclic Redundancy Check (CRC) :** Uses polynomial division to generate a checksum (called the CRC code). Highly reliable and widely used in networks and storage devices.
   - **Longitudinal Redundancy Check (LRC):** Similar to parity, but applies parity bits to a block of data instead of single bytes. Checks data along rows and columns, improving error detection.

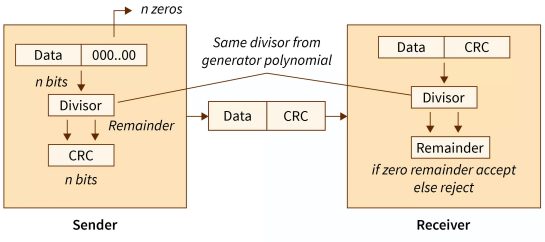
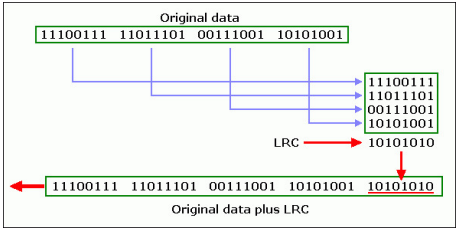


Figure: Cyclic Redundancy Check



1010*0*0*1* 0011*0*0*1* 11011101 11100111
10101010 (LRC)

Calculate the LRC for Data Received

1010*0*0*1*
0011*0*0*1*
11011101
11100111

→ LRC Calculated by Receiver 10101010
→ Compare with LRC Received 10101010

Figure: Longitudinal Redundancy Check

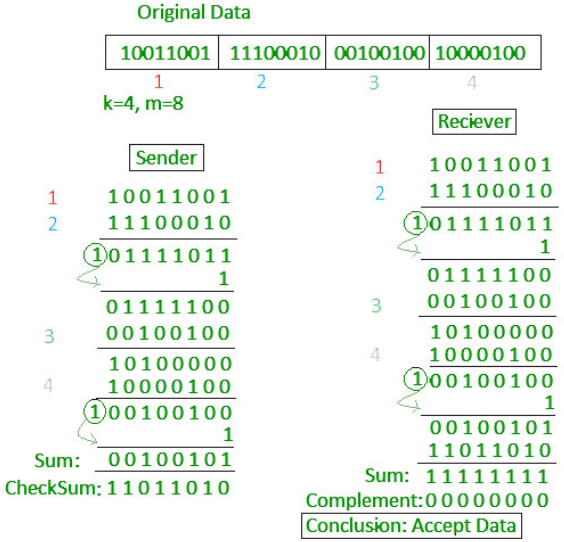
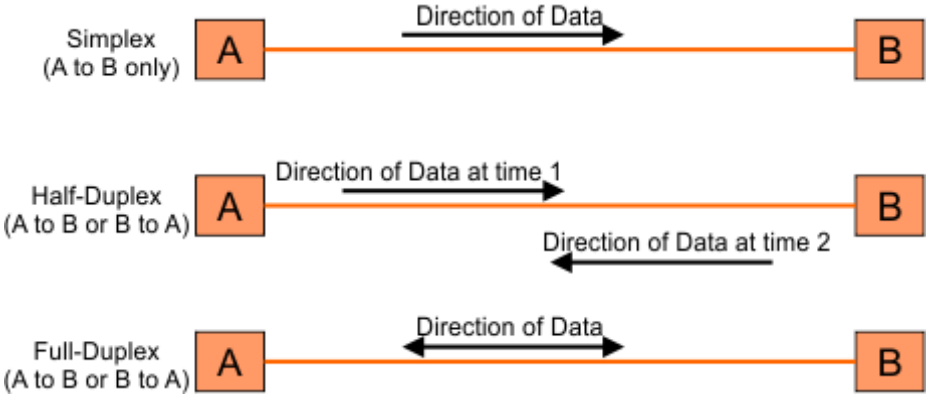


Figure: Checksum

3. Transmission Modes
   - **Simplex :** Data flows in only one direction, from sender to receiver. No return path. Example: Keyboard to computer.
   - **Half Duplex :** Data flows in both directions, but not simultaneously. Devices take turns transmitting. DMA (Direct Memory Access) Controller communication: The CPU and DMA controller take turns accessing the memory bus.
   - **Full Duplex :** Data flows simultaneously in both directions. Requires separate channels or techniques to manage bidirectional communication. Example: CPU and Cache Memory Communication where modern processors and caches communicate over separate lines for requests and responses, enabling simultaneous data transfer both ways.Network Interface Controller (NIC) inside a computer supporting full-duplex Ethernet communication where data can be sent and received simultaneously on separate channels.


Simplex
(A to B only)
A
Direction of Data
B
Half-Duplex
(A to B or B to A)
A
Direction of Data at time 1
B
Direction of Data at time 2
Full-Duplex
(A to B or B to A)
A
Direction of Data
B

Data Link & Protocol

- A Data Communication Processor (DCP) or Communication Interface Unit often implements data link functionalities to:
  - Offload the CPU from communication tasks.
  - Handle framing, error checking, and protocol-specific functions.
  - All these are associated with the Data Link.
- A protocol is a set of rules that define how data is formatted, transmitted, and received between systems.
  - Several Bit-Oriented Protocols, Byte-Oriented Protocols, Communication Protocols like TCP/IP, Point-to-Point Protocol, Ethernet Protocols (IEEE 802.3) etc.

Assignment

1. Compare and contrast between programmed I/O and interrupt driven I/O. Explain CPU and IOP communication channel using diagram. [5+5]
2. Why IOP is used in input-output organization? With the help of a neat diagram, explain how DMA technique is used to transfer data in a computer system. [3+7]
3. Elaborate the roles of I/O interface in a computer system. Explain how data transfer is performed with programmed I/O technique with necessary diagram. [4+6]
4. Differentiate between isolated and memory mapped Input-output. Explain with block diagram of DMA controller.[4+6]
5. Why data communication processor is required in an I/O organization? Why memory address spaces are reduced in memory mapped I/O ?[3+3]
6. Explain the types of priority interrupt in details.[10]
7. Explain different modes of Transfer in details. [10]
8. Explain block diagram of I/O module. [10]