• Peripheral Devices
• Input-Output Interface
• Asynchronous Data Transfer
• Modes of Transfer
• Priority Interrupt
• Direct Memory Access
**PERIPHERAL DEVICES**

### Input Devices
- Keyboard
- Optical input devices
  - Card Reader
  - Paper Tape Reader
  - Bar code reader
  - Digitizer
  - Optical Mark Reader
- Magnetic Input Devices
  - Magnetic Stripe Reader
- Screen Input Devices
  - Touch Screen
  - Light Pen
  - Mouse
- Analog Input Devices

### Output Devices
- CRT
- LCD
- Printer (Impact, Ink Jet, Laser, Dot Matrix)
- Plotter
- Analog
- Voice
**INPUT/OUTPUT INTERFACE**

- An I/O interface provides a way to transfer information between internal storage (such as memory and CPU registers) and external I/O devices.

- An I/O interface is necessary because of differences between the I/O device and the computer system:
  - The I/O device may use different standards for sending data than the computer system
  - I/O devices are typically slower than computer systems
  - Data representations may differ between the I/O device and the computer system
  - To provide a synchronization mechanism.

- As the name implies, an I/O interface is hardware that interfaces between the I/O device and the computer system
  - That is, it overcomes the differences between the two and allows them to communicate
I/O BUS AND INTERFACE MODULES

Each peripheral has an interface module associated with it.

Interface

- Decodes the device address (device code)
- Decodes the commands (operation)
- Provides signals for the peripheral controller
- Synchronizes the data flow and supervises the transfer rate between peripheral and CPU or Memory
I/O BUS AND MEMORY BUS

Functions of Buses

* **MEMORY BUS** is for *information transfers between CPU and the MM*

* **I/O BUS** is for information transfers between CPU and I/O devices through their I/O interface

Physical Organizations

* Some computer systems use two separate buses, one to communicate with memory and the other with I/O interfaces

* Many computers use a common single bus system for both memory and I/O interface units
  - Use one common bus but separate control lines for each function
  - Use one common bus with common control lines for both functions
ISOLATED vs MEMORY MAPPED I/O

Isolated I/O

- Separate I/O read/write control lines in addition to memory read/write control lines
- Separate (isolated) memory and I/O address spaces
- Distinct input and output instructions

Memory-mapped I/O

- A single set of read/write control lines (no distinction between memory and I/O transfer)
- Memory and I/O addresses share the common address space
  -> reduces memory address range available
- No specific input or output instruction
  -> The same memory reference instructions can be used for I/O transfers
- Considerable flexibility in handling I/O operations
Programmable Interface

- Information in each port can be assigned a meaning depending on the mode of operation of the I/O device
  → Port A = Data; Port B = Command; Port C = Status

- CPU initializes (loads) each port by transferring a byte to the Control Register
  → Allows CPU can define the mode of operation of each port
  → Programable Port: By changing the bits in the control register, it is possible to change the interface characteristics
ASYNCHRONOUS DATA TRANSFER

Synchronous and Asynchronous Operations

Synchronous - All devices derive the timing information from common clock line
Asynchronous - No common clock

Asynchronous Data Transfer

Asynchronous data transfer between two independent units requires that control signals be transmitted between the communicating units to indicate the time at which data is being transmitted

Two Asynchronous Data Transfer Methods

Strobe pulse
- A strobe pulse is supplied by one unit to indicate the other unit when the transfer has to occur

Handshaking
- A control signal is accompanied with each data being transmitted to indicate the presence of data
- The receiving unit responds with another control signal to acknowledge receipt of the data
STROBE CONTROL

* Employs a single control line to time each transfer
* The strobe may be activated by either the source or the destination unit

Source-Initiated Strobe for Data Transfer

**Block Diagram**

```
Source unit --- Data bus --- Destination unit
          |                   |
          Strobe          Strobe
```

**Timing Diagram**

Data --- Valid data

Destination-Initiated Strobe for Data Transfer

**Block Diagram**

```
Source unit --- Data bus --- Destination unit
          |                   |
          Strobe          Strobe
```

**Timing Diagram**

Data --- Valid data

Strobe

Asynchronous Data Transfer
Strobe Methods

Source-Initiated

The source unit that initiates the transfer has no way of knowing whether the destination unit has actually received data.

Destination-Initiated

The destination unit that initiates the transfer has no way of knowing whether the source has actually placed the data on the bus.

To solve this problem, the HANDSHAKE method introduces a second control signal to provide a Reply to the unit that initiates the transfer.
SOURCE-INITIATED TRANSFER USING HANDSHAKE

* Allows arbitrary delays from one state to the next
* Permits each unit to respond at its own data transfer rate
* The rate of transfer is determined by the slower unit
Destination-Initiated Transfer Using Handshake

Handshaking provides a high degree of flexibility and reliability because the successful completion of a data transfer relies on active participation by both units. If one unit is faulty, data transfer will not be completed. This can be detected by means of a *timeout* mechanism.
ASYNCHRONOUS SERIAL TRANSFER

Four Different Types of Transfer

<table>
<thead>
<tr>
<th>Asynchronous serial transfer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronous serial transfer</td>
</tr>
<tr>
<td>Asynchronous parallel transfer</td>
</tr>
<tr>
<td>Synchronous parallel transfer</td>
</tr>
</tbody>
</table>

Asynchronous Serial Transfer

- Employs special bits which are inserted at both ends of the character code
- Each character consists of three parts; Start bit; Data bits; Stop bits.

A character can be detected by the receiver from the knowledge of 4 rules:
- When data are not being sent, the line is kept in the 1-state (idle state)
- The initiation of a character transmission is detected by a Start Bit, which is always a 0
- The character bits always follow the Start Bit
- After the last character, a Stop Bit is detected when the line returns to the 1-state for at least 1 bit time

The receiver should know in advance the transfer rate of the bits and the number of information bits to expect.
A typical asynchronous communication interface available as an IC

Transmitter Register
- Accepts a data byte (from CPU) through the data bus
- Transferred to a shift register for serial transmission

Receiver
- Receives serial information into another shift register
- Complete data byte is sent to the receiver register

Status Register Bits
- Used for I/O flags and for recording errors

Control Register Bits
- Define baud rate, no. of bits in each character, whether to generate and check parity, and no. of stop bits

---

### Asynchronous Data Transfer

<table>
<thead>
<tr>
<th>CS</th>
<th>RS</th>
<th>Oper.</th>
<th>Register selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x</td>
<td>x</td>
<td>None</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>WR</td>
<td>Transmitter register</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>WR</td>
<td>Control register</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>RD</td>
<td>Receiver register</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>RD</td>
<td>Status register</td>
</tr>
</tbody>
</table>
3 different Data Transfer Modes between the central computer (CPU or Memory) and peripherals:

Program-Controlled I/O (Input Dev to CPU)

- **Polling or Status Checking**
  - Continuous CPU involvement
  - CPU slowed down to I/O speed
  - Simple
  - Least hardware

- **Direct Memory Access (DMA)**

- **Interrupt-Initiated I/O**
Interrupt Initiated I/O

- Polling takes valuable CPU time
- Open communication only when some data has to be passed -> *Interrupt*.
- I/O interface, instead of the CPU, monitors the I/O device
- When the interface determines that the I/O device is ready for data transfer, it generates an *Interrupt Request* to the CPU
- Upon detecting an interrupt, CPU stops momentarily the task it is doing, branches to the service routine to process the data transfer, and then returns to the task it was performing

DMA (Direct Memory Access)

- Large blocks of data transferred at a high speed to or from high speed devices, magnetic drums, disks, tapes, etc.
- DMA controller
  
  Interface that provides I/O transfer of data directly to and from the memory and the I/O device
- CPU initializes the DMA controller by sending a memory address and the number of words to be transferred
- Actual transfer of data is done directly between the device and memory through DMA controller
  -> Freeing CPU for other tasks
PRIORITY INTERRUPT

Priority
- 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)

Priority Interrupt by Software (Polling)
- Priority is established by the order of polling the devices (interrupt sources)
- Flexible since it is established by software
- Low cost since it needs a very little hardware
- Very slow

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
HARDWARE PRIORITY INTERRUPT - DAISY-CHAIN -

Interrupt Request from any device(>=1)
- CPU responds by INTACK <- 1
- Any device receives signal(INTACK) 1 at PI puts the VAD on the bus
Among interrupt requesting devices the only device which is physically closest to CPU gets INTACK=1, and it blocks INTACK to propagate to the next device

One stage of the daisy chain priority arrangement

<table>
<thead>
<tr>
<th>PI</th>
<th>RF</th>
<th>PO</th>
<th>Enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
PARALLEL PRIORITY INTERRUPT

IEN: Set or Clear by instructions ION or IOF
IST: Represents an unmasked interrupt has occurred. INTACK enables tristate Bus Buffer to load VAD generated by the Priority Logic

Interrupt Register:
- Each bit is associated with an Interrupt Request from different Interrupt Source - different priority level
- Each bit can be cleared by a program instruction

Mask Register:
- Mask Register is associated with Interrupt Register
- Each bit can be set or cleared by an Instruction
INTERRUPT PRIORITY ENCODER

Determines the highest priority interrupt when more than one interrupts take place

Priority Encoder Truth table

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
<th>Boolean functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>I₀  I₁ I₂ I₃</td>
<td>x  y  IST</td>
<td>x = I₀' I₁'</td>
</tr>
<tr>
<td>1  d  d  d</td>
<td>0  0  1</td>
<td>y = I₀' I₁ + I₀' I₂'</td>
</tr>
<tr>
<td>0  1  d  d</td>
<td>0  1  1</td>
<td>(IST) = I₀ + I₁ + I₂ + I₃</td>
</tr>
<tr>
<td>0  0  1  d</td>
<td>1  0  1</td>
<td></td>
</tr>
<tr>
<td>0  0  0  1</td>
<td>1  1  1</td>
<td></td>
</tr>
<tr>
<td>0  0  0  0</td>
<td>d  d  0</td>
<td></td>
</tr>
</tbody>
</table>
At the end of each Instruction cycle

- CPU checks IEN and IST
- If \( IEN \cdot IST = 1 \), CPU -> Interrupt Cycle

\[
\begin{align*}
\text{SP} & \leftarrow \text{SP} - 1 \quad \text{Decrement stack pointer} \\
\text{M}[\text{SP}] & \leftarrow \text{PC} \quad \text{Push PC into stack} \\
\text{INTACK} & \leftarrow 1 \quad \text{Enable interrupt acknowledge} \\
\text{PC} & \leftarrow \text{VAD} \quad \text{Transfer vector address to PC} \\
\text{IEN} & \leftarrow 0 \quad \text{Disable further interrupts} \\
\text{Go To Fetch} & \quad \text{to execute the first instruction in the interrupt service routine}
\end{align*}
\]
Initial and Final Operations

Each interrupt service routine must have an initial and final set of operations for controlling the registers in the hardware interrupt system.

**Initial Sequence**
1. Clear lower level Mask reg. bits
2. IST <- 0
3. Save contents of CPU registers
4. IEN <- 1
5. Go to Interrupt Service Routine

**Final Sequence**
1. IEN <- 0
2. Restore CPU registers
3. Clear the bit in the Interrupt Reg
4. Set lower level Mask reg. bits
5. Restore return address, IEN <- 1
**DIRECT MEMORY ACCESS**

* Block of data transfer from high speed devices, Drum, Disk, Tape
* DMA controller - Interface which allows I/O transfer directly between Memory and Device, freeing CPU for other tasks
* CPU initializes DMA Controller by sending memory address and the block size (number of words)

**CPU bus signals for DMA transfer**

**Block diagram of DMA controller**

- Address bus
- Data bus
- DMA select
- Register select
- Read
- Write
- Bus request
- Bus grant
- Interrupt

- Control logic
- Data bus buffers
- Address bus buffers
- Address register
- Word count register
- Control register
- DMA request
- DMA acknowledge
- to I/O device

**High-impedence (disabled) when BG is enabled**
DMA I/O OPERATION

Starting an I/O
- CPU executes instruction to
  Load Memory Address Register
  Load Word Counter
  Load Function (Read or Write) to be performed
  Issue a GO command

Upon receiving a GO Command DMA performs I/O operation as follows independently from CPU

Input
1. Input Device <- R (Read control signal)
2. Buffer (DMA Controller) <- Input Byte; and
   assembles the byte into a word until word is full
3. M <- memory address, W (Write control signal)
4. Address Reg <- Address Reg + 1; WC (Word Counter) <- WC - 1
5. If WC = 0, then Interrupt to acknowledge done, else go to [1]

Output
1. M <- M Address, R
   M Address R <- M Address R + 1, WC <- WC - 1
2. Disassemble the word
3. Buffer <- One byte; Output Device <- W, for all disassembled bytes
4. If WC = 0, then Interrupt to acknowledge done, else go to [1]
CYCLE STEALING

While DMA I/O takes place, CPU is also executing instructions

DMA Controller and CPU both access Memory -> Memory Access Conflict

Memory Bus Controller
- Coordinating the activities of all devices requesting memory access
- Priority System

Memory accesses by CPU and DMA Controller are interwoven, with the top priority given to DMA Controller
-> Cycle Stealing

Cycle Steal
- CPU is usually much faster than I/O(DMA), thus CPU uses the most of the memory cycles
- DMA Controller steals the memory cycles from CPU
- For those stolen cycles, CPU remains idle
Direct Memory Access

DMA TRANSFER

- DMA request
- DMA ack.
- I/O Peripheral device
- DMA request

CPU

RD WR Addr Data

Random-access memory unit (RAM)

RD WR Addr Data

DMA Controller

RD WR Addr Data

Interrupt
BG BR

Address select

Read control
Write control

Data bus
Address bus

I/O Peripheral device