UNIT – V
Interfacing and Communication

I/O

For the computer to be of use to us it must be able to obtain data from the outside world, process it (this is the part we have been discussing), an then communicate the information back to the outside world.

How does the CPU implement input and output transactions?

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

- Card Puncher, Paper Tape Puncher
- CRT
- Printer (Impact, Ink Jet, Laser, Dot Matrix)
- Plotter
- Analog
- Voice

External Devices

Provide a means of exchanging data between the external environment and the computer

Attach to the computer by a link to an I/O module

- The link is used to exchange control, status, and data between the I/O module and the external device

peripheral device

- An external device connected to an I/O module

Three categories:

Human readable

- Suitable for communicating with the computer user
- Video display terminals (VDTs), printers

Machine readable

- Suitable for communicating with equipment
- Magnetic disk and tape systems, sensors and actuators

Communication

- Suitable for communicating with remote devices such as a terminal, a machine readable device, or another computer

External Device Block Diagram

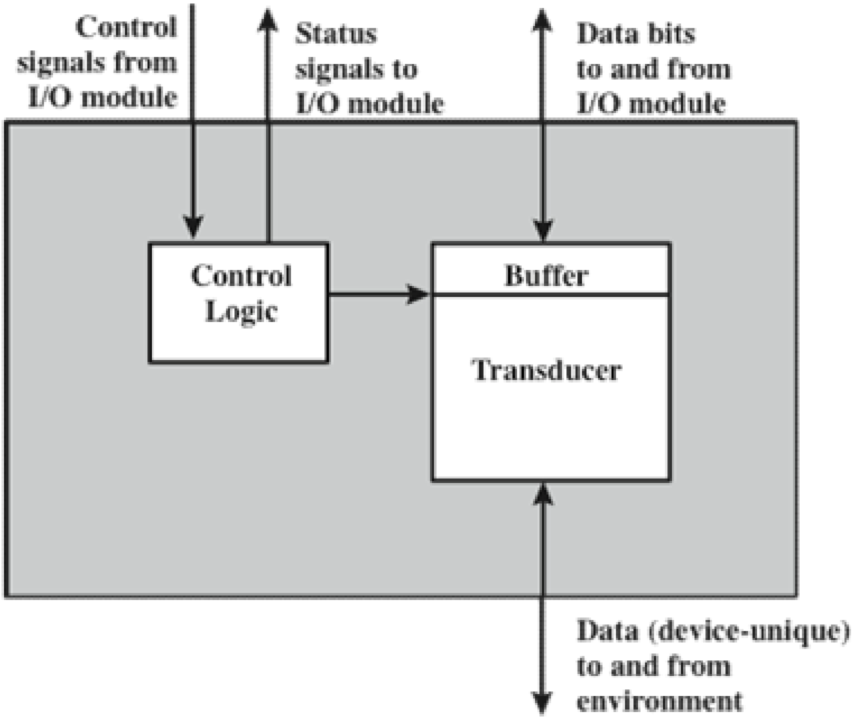

Control signals from I/O module
Status signals to I/O module
Data bits to and from I/O module
Control Logic
Buffer
Transducer
Data (device-unique) to and from environment

Figure 7.2 Block Diagram of an External Device

A partial answer is it doesn't do all of the work itself.

Instead the fine details of I/O are handled by special purpose interface chips.

These chips handle the transfer of data to the external device which then does further processing before the information is presented to the end user.

Input / Output Fundamentals

We divide I/O into three areas:

1. Strategy

2. Interface circuitry

3. I/O device – the peripheral.

INPUT/OUTPUT INTERFACES

* Provides a method for transferring information between internal storage (such as memory and CPU registers) and external I/O devices

* Resolves the *differences* between the computer and peripheral devices

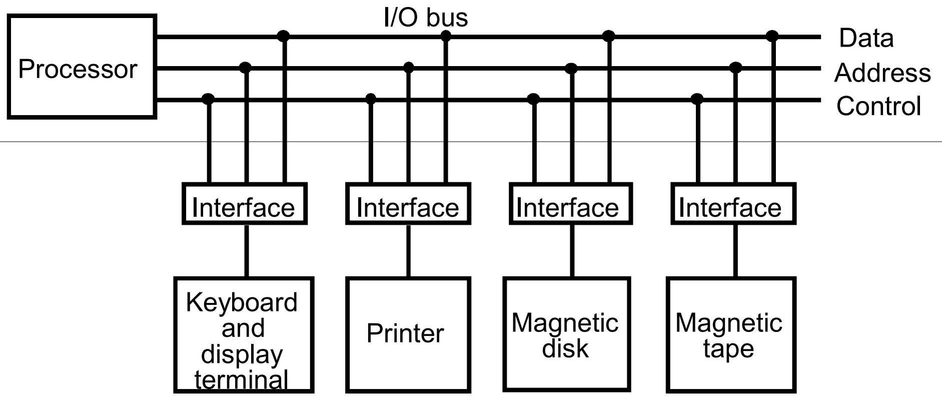
- Peripherals - Electromechanical Devices
  CPU or Memory - Electronic Device

  - Data Transfer Rate
    Peripherals - Usually slower
    CPU or Memory - Usually faster than peripherals
    Some kinds of Synchronization mechanism may be needed

  - Unit of Information
    Peripherals - Byte
    CPU or Memory - Word

  - Operating Modes
    Peripherals - Autonomous, Asynchronous
    CPU or Memory - Synchronous

I/O BUS AND INTERFACE MODULES



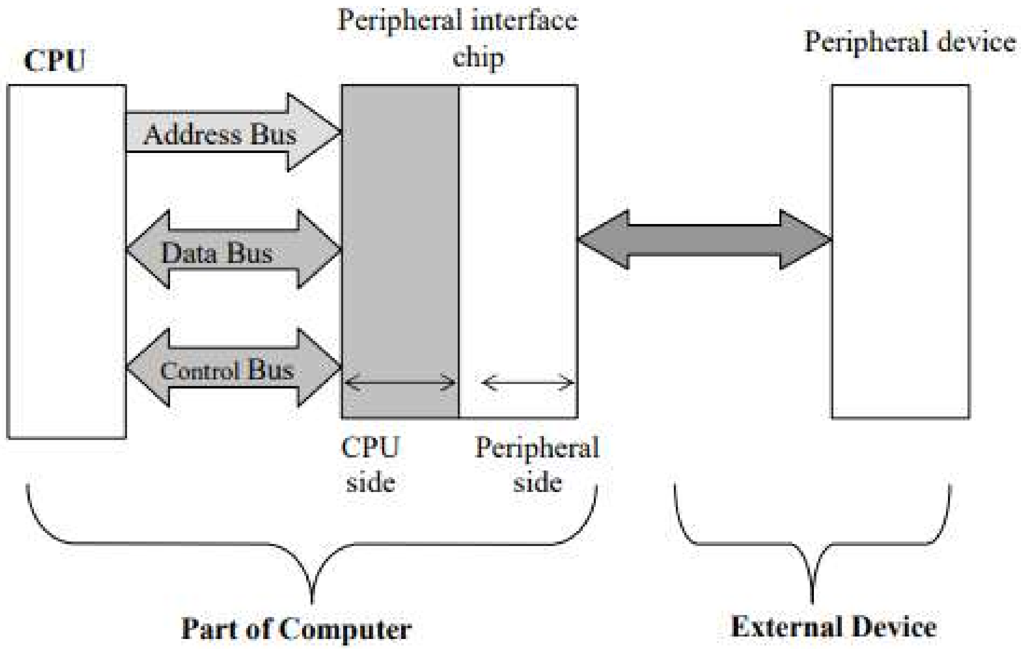
Each peripheral has an interface module associated with it

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

Typical I/O instruction

Op. code	Device address	Function code
		(Command)


CPU
Peripheral interface chip
Peripheral device
Address Bus
Data Bus
Control Bus
CPU side
Peripheral side
Part of Computer
External Device

I/O Module Structure

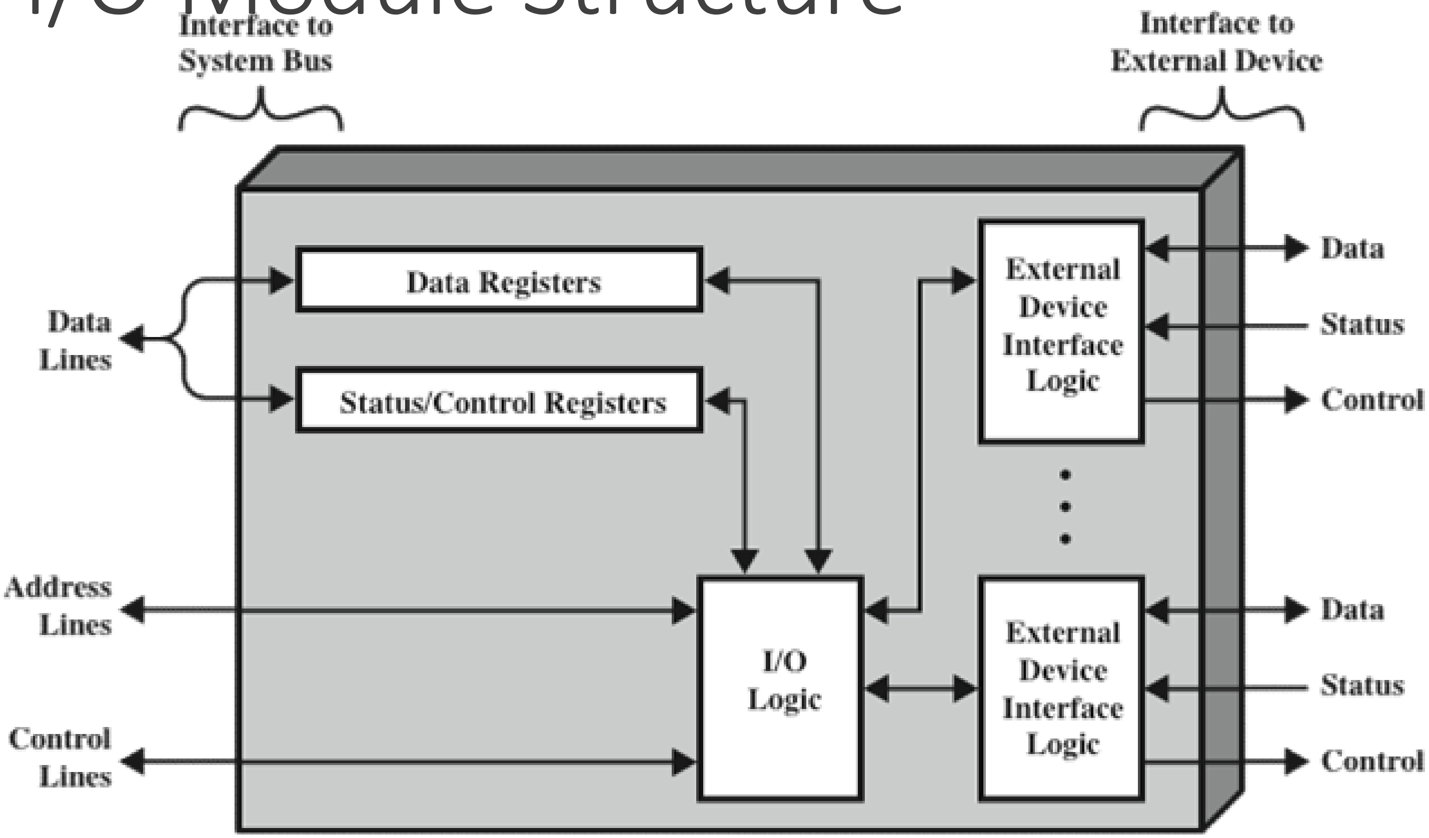
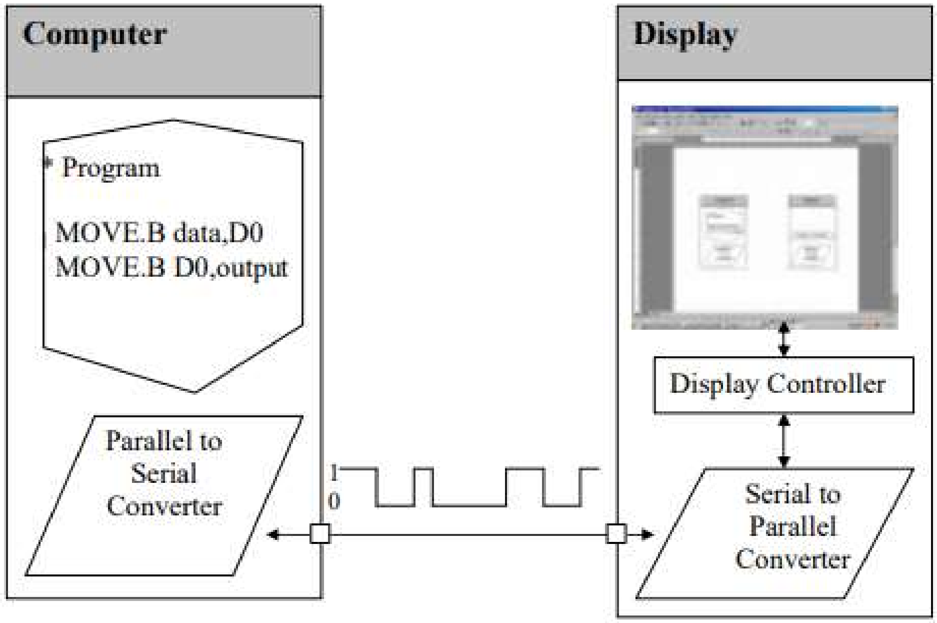


Figure 7.3 Block Diagram of an I/O Module

Relationship between computer and peripheral



```
MOVE.B data,D0
MOVE.B D0,output
```

Handshaking and buffering

The process of two devices communicating is itself a complex task. It must involve making sure the devices are indeed communicating with each other and must take into account the vast differences in the speed between devices.

All data transfers can be grouped into one of two classes:

- Open loop

Data is sent and reception is assumed

- Closed loop

Data is sent and an acknowledgement of receipt is returned.

Handshaking

Many devices will communicate information back to the CPU indicating they have received data or are ready to receive data. The device that initiates the data transfer is often referred to as the master and the device addressed by the master is often called the slave.

The master and slave communicate via a sequence of signals indicating data ready, data accepted, data no longer valid. The sequence of communications is called *handshaking.*

Handshaking can be used to deal with the difference in speed of the master and slave.

Handshaking

The handshaking process can be extended to have the devices acknowledge the de-assertion of signals (we refer to this strict sequence of steps as **delay insensitive** since a new step can not take place without the proper acknowledgements of activations and deactivations of signals).

We will call such a data transfer a ***fully interlocked*** data transfer.

Buffering

Buffered I/O

A typical problem with input and output is devices tend not to operate at optimal speeds. It is often the case that data items arrive faster than they can be processed. Rather than ignore or loose the data it can be loaded into a special device called FIFO memory (First In First Out). As data arrives at the input port it is stored in order of arrival. The designer of the port is responsible for supplying sufficient memory to handle worse-case input bursts.

The process of saving data in a store until it is needed is called **buffering**.

Buffering

FIFO – the **data structure** used to implement a FIFO is called a queue.

Typically a queue will grow from lower addresses to higher addresses. New data are added at higher addresses while data is retrieved at the lower addresses. Thus as described so far, a queue would tend to move through

Mode of Data Transfer

Data transfer to and from the peripherals may be done in any of the three possible ways:

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.

- The processor executes a program that gives it direct control of the I/O operation, including sensing device status, sending a read or write command, and transferring the data.

- When the processor issues a command to the I/O module, it must wait until the I/O operation is complete. If the processor is faster than the I/O module, this is wasteful of processor time.

The overall operation of the programmed I/O can be summaries as follow:

The processor is executing a program and encounters an instruction relating to I/O operation.

The processor then executes that instruction by issuing a command to the appropriate I/O module.

The I/O module will perform the requested action based on the I/O command issued by the processor (READ/WRITE) and set the appropriate bits in the I/O status register.

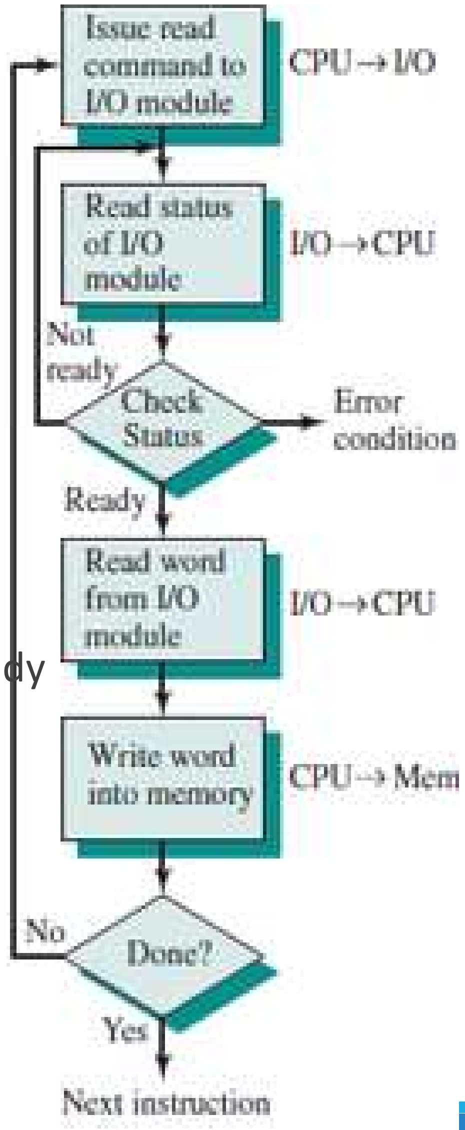
The processor will periodically check the status of the I/O module until it find that the operation is complete.

Programmed I/O *Mode*
Input Data Transfer

- Each input is read after first testing whether the device is ready with the input (a state reflected by a bit in a status register).

- The program waits for the ready status by repeatedly testing the status bit and till all targeted bytes are read from the input device.

- The program is in busy (non-waiting) state only after the device gets ready else in wait state.

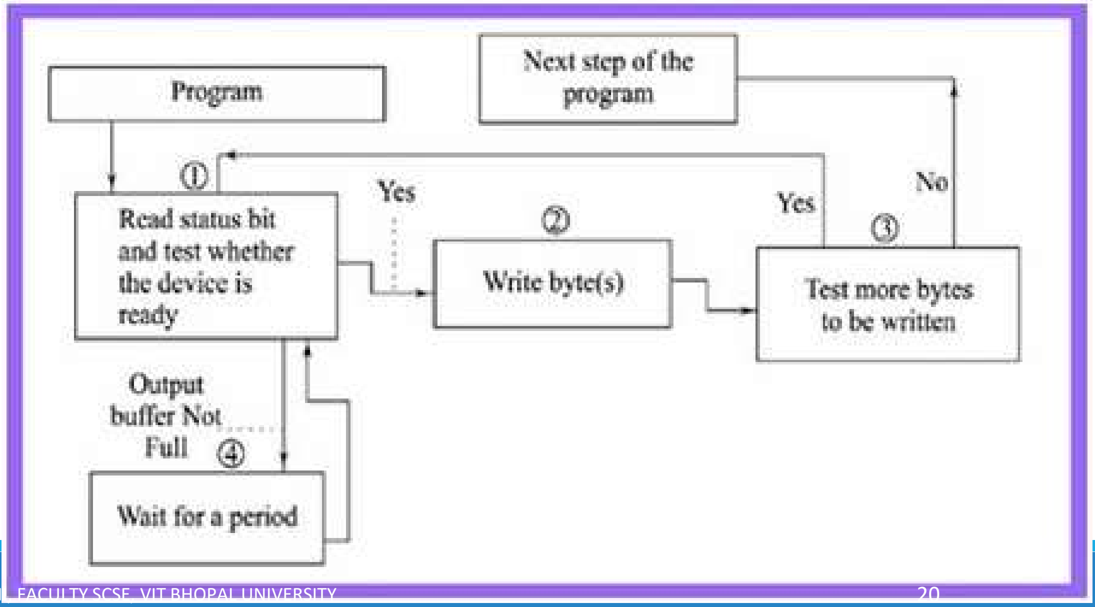


Programmed I/O Mode Output Data Transfer

Each output written after first testing whether the device is ready to accept the byte at its output register or output buffer is empty.

The program waits for the ready status by repeatedly testing the status bit(s) and till all the targeted bytes are written to the device.

The program in busy (non-waiting) state only after the device gets ready else wait state.



Example

Example: Consider the example of a computer connected to a CRT and a Keyboard.

Input and output is done via a strategy called **Programmed Data Transfer** –

- when we want data an instruction reads data from the input port the keyboard is connected to.
- When the computer wishes to display information data is written to the output port that communicates with the CRT.

A port is thought of as a gateway between the computer and an external device.

Typically data is written from CPU to the output port in parallel form at 8/16/32 bits at a time.

The output port must serialize the information and transmit it bit by bit over the twisted pair to the CRT. In addition, the CRT and output port must synchronize their communications.

The output device is the CRT. It accepts a form of serial communication & needs to reconstruct a parallel signal used , for instance, to select a character from a table of symbols.

Advantages & Disadvantages of Programmed I/O

Advantages	- simple to implement
	- very little hardware support
Disadvantages	- busy waiting
	- ties up CPU for long period with no useful work

Interrupt- initiated or Interrupt Driven I/O

Since in the above case we saw the CPU is kept busy unnecessarily. This situation can very well be avoided by using an interrupt driven method for data transfer.

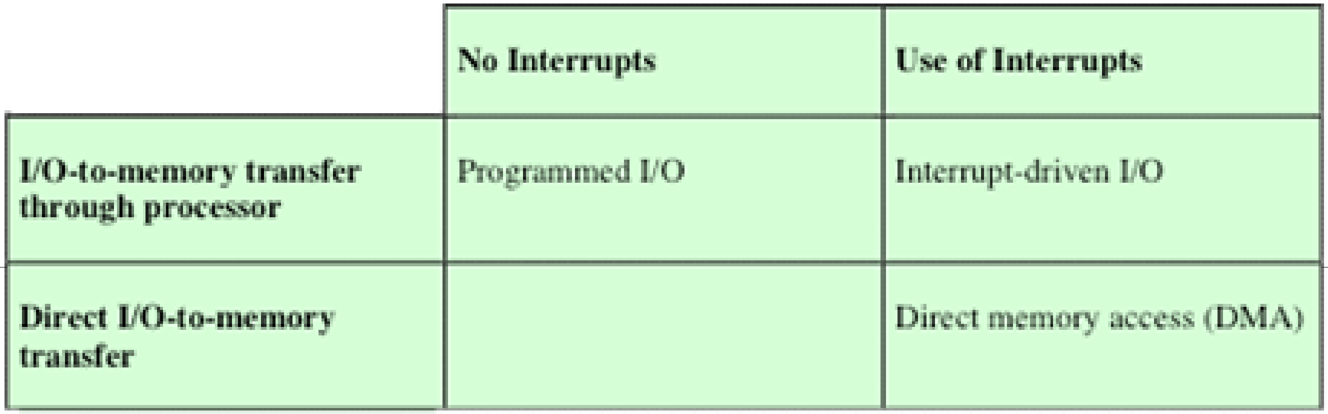
By using interrupt facility and special commands to inform the interface to issue an interrupt request signal whenever data is available from any device. In the meantime the CPU can proceed for any other program execution.

The interface meanwhile keeps monitoring the device. Whenever it is determined that the device is ready for data transfer it initiates an interrupt request signal to the computer.

Upon detection of an external interrupt signal the CPU stops momentarily the task that it was already performing, branches to the service program to process the I/O transfer, and then return to the task it was originally performing.

Note: Both the methods programmed I/O and Interrupt-driven I/O require the active intervention of the processor to transfer data between memory and the I/O module, and any data transfer must transverse a path through the processor. Thus both these forms of I/O suffer from two inherent drawbacks.

- The I/O transfer rate is limited by the speed with which the processor can test and service a device.
- The processor is tied up in managing an I/O transfer; a number of instructions must be executed for each I/O transfer.



	No Interrupts	Use of Interrupts
I/O-to-memory transfer through processor	Programmed I/O	Interrupt-driven I/O
Direct I/O-to-memory transfer		Direct memory access (DMA)

A second example: consider a block of data needing to be written to a disk drive. Again we start by looking at the strategy.

Programmed data transfer is probably not the scheme used. There is no real need to load the data from memory into the CPU registers and then write to a port.

Instead what is often used is a Direct Memory Access (DMA) strategy.

DMA: Direct Memory Access

The CPU instructs special purpose DMA hardware to move the block of memory, thereby freeing the CPU to continue with its calculations. The interface circuit is a chip called a DMA Controller (DMAC).

The DMAC is responsible for:

- providing the addresses for source or destination of data,
- signalling the peripheral data is needed or data is ready,
- coordinating with the CPU for control of data and address buses during the data transfer.
- The peripheral, the disk drive is a complex mixture of electronic and high precision mechanical devices. Data is stored by affecting magnetic properties of the surface of a high-speed rotating disk.

Typical DMA Module Diagram

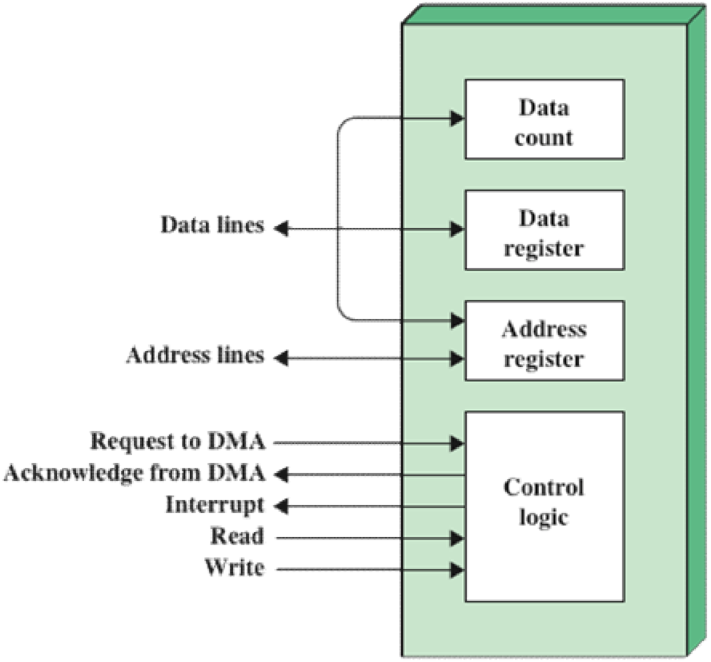


Figure 7.11 Typical DMA Block Diagram

Bus Request : It is used by the DMA controller to request the CPU to relinquish the control of the buses.

Bus Grant : It is activated by the CPU to Inform the external DMA controller that the buses are in high impedance state and the requesting DMA can take control of the buses. Once the DMA has taken the control of the buses it transfers the data. This transfer can take place in many ways.

Types of DMA transfer using DMA controller:

Burst Transfer :
DMA returns the bus after complete data transfer. A register is used as a byte count,
being decremented for each byte transfer, and upon the byte count reaching zero, the DMAC will release the bus. When the DMAC operates in burst mode, the CPU is halted for the duration of the data
transfer.
Steps involved are:

- Bus grant request time.
- Transfer the entire block of data at transfer rate of device because the device is usually slow than the speed at which the data can be transferred to CPU.
- Release the control of the bus back to CPU
  So, total time taken to transfer the N bytes
  = Bus grant request time + (N) * (memory transfer rate) + Bus release control time.

Cyclic Stealing :

In this DMA controller transfers one word at a time after which it must return the control of the buses to the CPU. The CPU merely delays its operation for one memory cycle to allow the direct memory I/O transfer to “steal” one memory cycle.
Steps Involved are:

- Buffer the byte into the buffer
- Inform the CPU that the device has 1 byte to transfer (i.e. bus grant request)
- Transfer the byte (at system bus speed)
- Release the control of the bus back to CPU.

Before moving on transfer next byte of data, device performs step 1 again so that bus isn’t tied up and the transfer won’t depend upon the transfer rate of device.
So, for 1 byte of transfer of data, time taken by using cycle stealing mode (T).
= time required for bus grant + 1 bus cycle to transfer data + time required to release the bus, it will be
N x T

Cyclic Stealing :

In cycle stealing mode we always follow pipelining concept that when one byte is getting transferred then Device is parallel preparing the next byte. “The fraction of CPU time to the data transfer time” if asked then cycle stealing mode is used.

Interleaved mode: In this technique , the DMA controller takes over the system bus when the microprocessor is not using it. An alternate half cycle i.e. half cycle DMA + half cycle processor.

Interrupts in 8085 microprocessor

When microprocessor receives any interrupt signal from peripheral(s) which are requesting its services, it stops its current execution and program control is transferred to a sub-routine by generating **CALL** signal and after executing sub-routine by generating **RET** signal again program control is transferred to main program from where it had stopped.

When microprocessor receives interrupt signals, it sends an acknowledgement (INTA) to the peripheral which is requesting for its service.

Interrupts can be classified into various categories based on different parameters:

DMA Operation

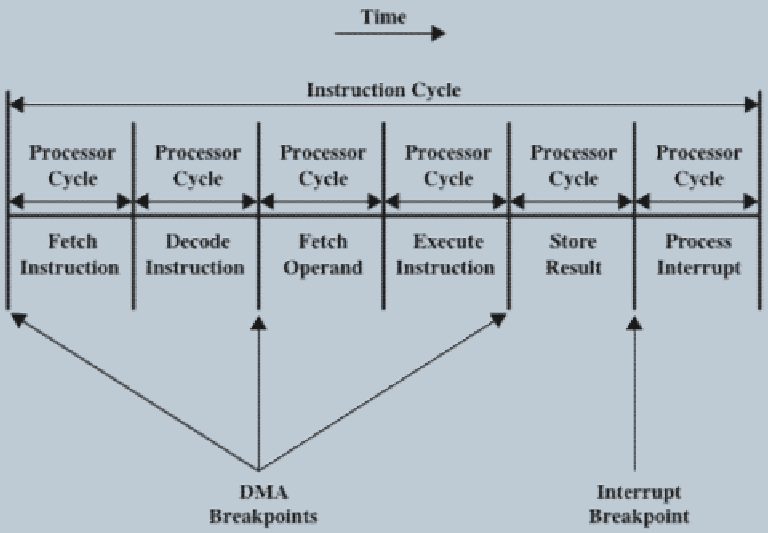


Figure 7.12 DMA and Interrupt Breakpoints During an Instruction Cycle

DMA

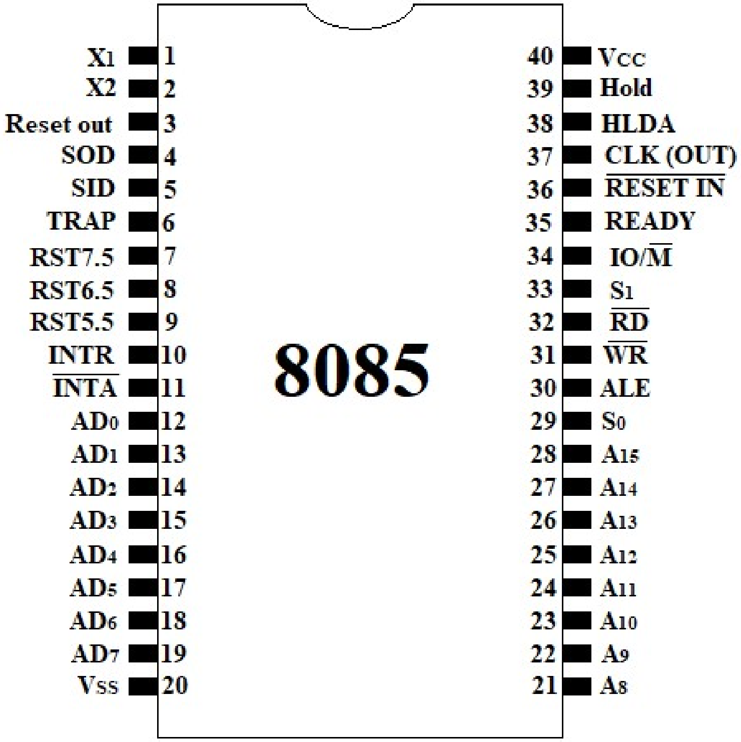
DMA

Interrupts in 8085 microprocessor

When microprocessor receives any interrupt signal from peripheral(s) which are requesting its services, it stops its current execution and program control is transferred to a sub-routine by generating **CALL** signal and after executing sub-routine by generating **RET** signal again program control is transferred to main program from where it had stopped.

When microprocessor receives interrupt signals, it sends an acknowledgement (INTA) to the peripheral which is requesting for its service.

Interrupts can be classified into various categories based on different parameters:


8085
X1 1
X2 2
Reset out 3
SOD 4
SID 5
TRAP 6
RST7.5 7
RST6.5 8
RST5.5 9
INTR 10
INTA 11
AD0 12
AD1 13
AD2 14
AD3 15
AD4 16
AD5 17
AD6 18
AD7 19
Vss 20
40 Vcc
39 Hold
38 HLDA
37 CLK (OUT)
36 RESET IN
35 READY
34 IO/M
33 S1
32 RD
31 WR
30 ALE
29 S0
28 A15
27 A14
26 A13
25 A12
24 A11
23 A10
22 A9
21 A8

Hardware and Software Interrupts

When microprocessors receive interrupt signals through pins (hardware) of microprocessor, they are known as *Hardware Interrupts*. There are 5 Hardware Interrupts in 8085 microprocessor. They are – *INTR, RST 7.5, RST 6.5, RST 5.5, TRAP*

Software Interrupts are those which are inserted in between the program which means these are mnemonics of microprocessor. There are 8 software interrupts in 8085 microprocessor. They are – *RST 0, RST 1, RST 2, RST 3, RST 4, RST 5, RST 6, RST 7*.

Vectored and Non-Vectored Interrupts

Vectored Interrupts are those which have fixed vector address (starting address of sub-routine) and after executing these, program control is transferred to that address.
Vector Addresses are calculated by the formula 8 * TYPE

INTERRUPT	VECTOR ADDRESS
TRAP (RST 4.5)	24 H
RST 5.5	2C H
RST 6.5	34 H
RST 7.5	3C H

For Software interrupts vector addresses are given by:

INTERRUPT	VECTOR ADDRESS
RST 0	00 H
RST 1	08 H
RST 2	10 H
RST 3	18 H
RST 4	20 H
RST 5	28 H
RST 6	30 H
RST 7	38 H

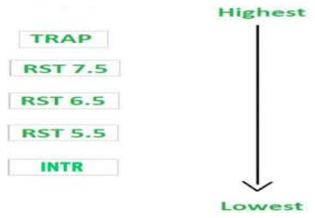
Non-Vectored Interrupts are those in which vector address is not predefined. The interrupting device gives the address of sub-routine for these interrupts. *INTR* is the only non-vectored interrupt in 8085 microprocessor.

Maskable and Non-Maskable Interrupts –

Maskable Interrupts are those which can be disabled or ignored by the microprocessor. These interrupts are either edge-triggered or level-triggered, so they can be disabled. *INTR, RST 7.5, RST 6.5, RST 5.5* are maskable interrupts in 8085 microprocessor.

Non-Maskable Interrupts are those which cannot be disabled or ignored by microprocessor. *TRAP* is a non-maskable interrupt. It consists of both level as well as edge triggering and is used in critical power failure conditions.

Priority of Interrupts –
When microprocessor receives multiple interrupt requests simultaneously, it will execute the interrupt service request (ISR) according to the priority of the interrupts.



Instruction for Interrupts –

Enable Interrupt (EI) – The interrupt enable flip-flop is set and all interrupts are enabled following the execution of next instruction followed by EI. No flags are affected. After a system reset, the interrupt enable flip-flop is reset, thus disabling the interrupts. This instruction is necessary to enable the interrupts again (except TRAP).

Disable Interrupt (DI) – This instruction is used to reset the value of enable flip-flop hence disabling all the interrupts. No flags are affected by this instruction.

Set Interrupt Mask (SIM) – It is used to implement the hardware interrupts (RST 7.5, RST 6.5, RST 5.5) by setting various bits to form masks or generate output data via the Serial Output Data (SOD) line. First the required value is loaded in accumulator then SIM will take the bit pattern from it.

Buses

Next presentation