Unit 4-Basic I/O

I/O Overview

- One of the basic features of a computer is its ability to exchange data with other devices.
- They are an integral part of home appliances, manufacturing equipment, transportation systems, banking, and point-of-sale terminals.
- In such applications, input to a computer may come from a sensor switch, a digital camera, a microphone, or a fire alarm.
- Output may be a sound signal sent to a speaker, or a digitally coded command that changes the speed of a motor, opens a valve, or causes a robot to move in a specified manner.

I/O module

- 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.

I/O module..

Why one does not connect peripherals directly to the system bus?

The reasons are as follows:

- 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.
- 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.
- 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
- Peripherals often use different data formats and word lengths than the computer to which they are attached.

I/O module..

Two major functions of this module:

- Interface to the processor and memory via the system bus or central switch.
- Interface to one or more peripheral devices by tailored data links.

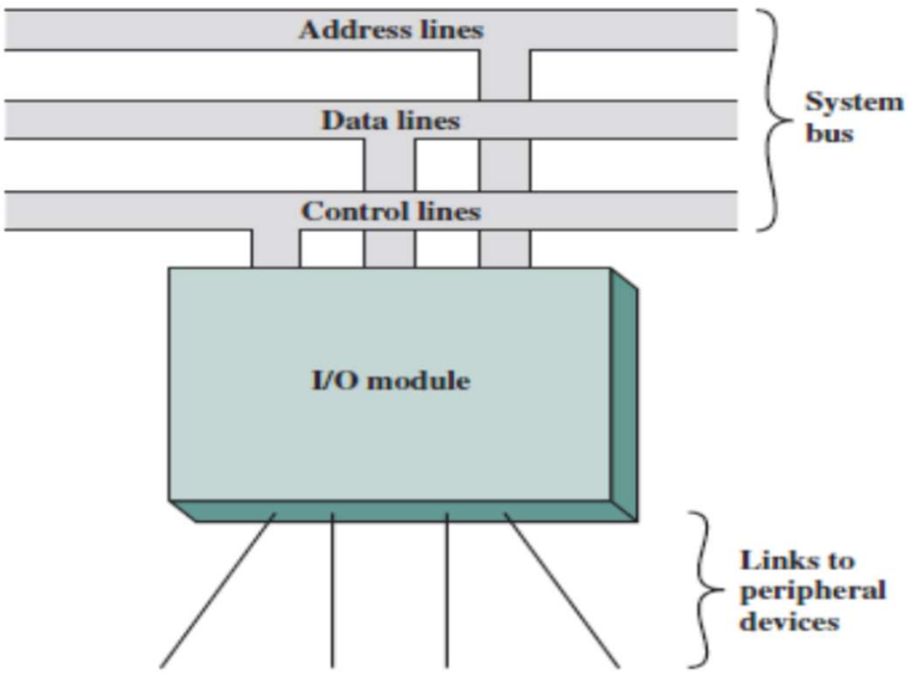


Figure 4.1 Generic Model of I/O Module

External devices

- An external device attaches 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.
- An external device connected to an I/O module is often referred to as a peripheral device or, simply, a peripheral.

Classification external devices:

- Human-computer communication
- Computer-computer communication
- Computer-device communication

Block Diagram of an External Device

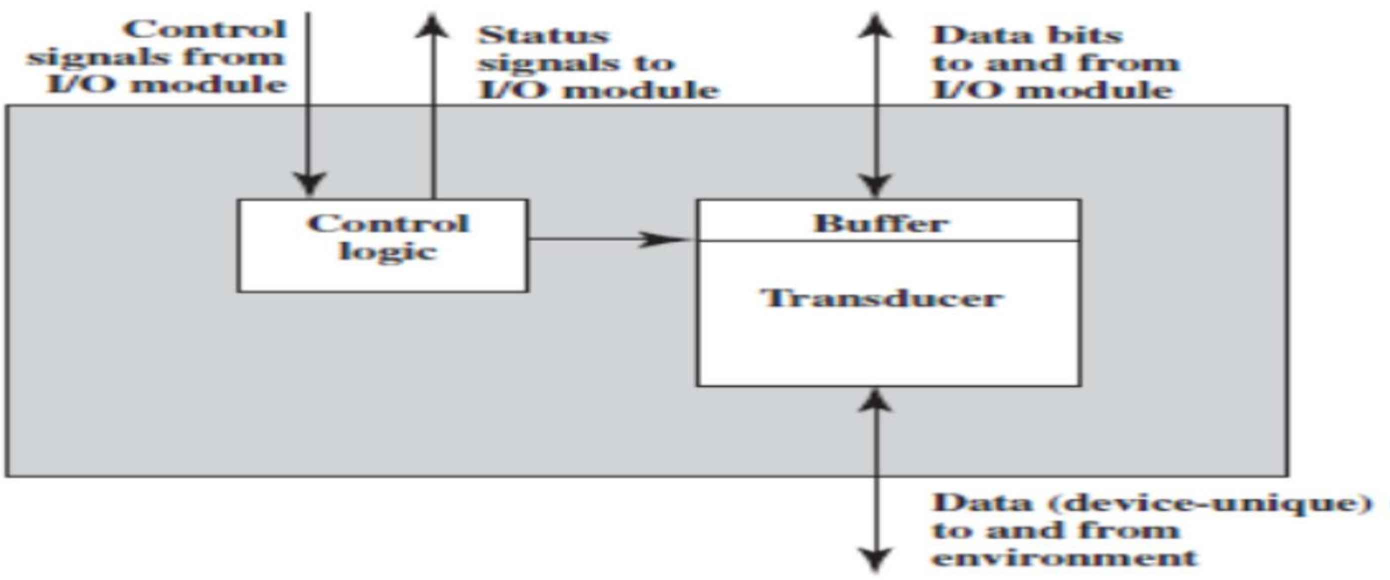


Figure 4.2 Block Diagram of External Device

Block Diagram of an External Device..

- 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.
- Data are in the form of a set of bits to be sent to or received from the I/O module.
- Status signals indicate the state of the device.
  - Examples are READY/ NOT-READY to show whether the device is ready for data transfer.

Block Diagram of an External Device..

- Control logic associated with the device controls the device’s operation in response to direction 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.

I/O Modules functions

The major functions or requirements for an I/O module fall into the following categories:

- Control and timing
- Processor communication
- Device communication
- Data buffering
- Error detection

I/O module functions..

- The internal resources, such as main memory and the system bus, must be shared among a number of activities, including data I/O.
- Thus, the I/O function includes a control and timing requirement, to coordinate the flow of traffic between internal resources and external devices.

I/O module functions..

The control of the transfer of data from an external device to the processor might involve the following sequence of steps:

1. The processor interrogates the I/O module to check the status of the attached device.
2. The I/O module returns the device status.
3. 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.
4. The I/O module obtains a unit of data (e.g., 8 or 16 bits) from the external device.
5. The data are transferred from the I/O module to the processor.

I/O module functions..

Processor Communication involves the following:

- **Command decoding:** The I/O module accepts commands from the processor, typically sent as signals on the control bus.
  - For example, an I/O module for a disk drive might accept the following commands: READ SECTOR, WRITE SECTOR, SEEK track number, and SCAN record ID.
- **Data**:Data are exchanged between the processor and the I/O module over the data bus.
- **Status reporting**: Because peripherals are so slow, it is important to know the status of the I/O module.
  - Common status signals are BUSY and READY. There may also be signals to report various error conditions.
- **Address recognition**: I/O module must recognize one unique address for each peripheral it controls.

I/O module functions..

- An essential task of an I/O module is **data buffering**.
- The transfer rate into and out of main memory or the processor is quite high, the rate is orders of magnitude lower for many peripheral devices and covers a wide range.
- Data coming from main memory are sent to an I/O module in a rapid burst.
- The data are buffered in the I/O module and then sent to the peripheral device at its data rate.
- I/O module is often responsible for **error detection** and for subsequently reporting errors to the processor.
  - One class of errors includes mechanical and electrical malfunctions reported by the device (e.g., paper jam, bad disk track)

Computer system

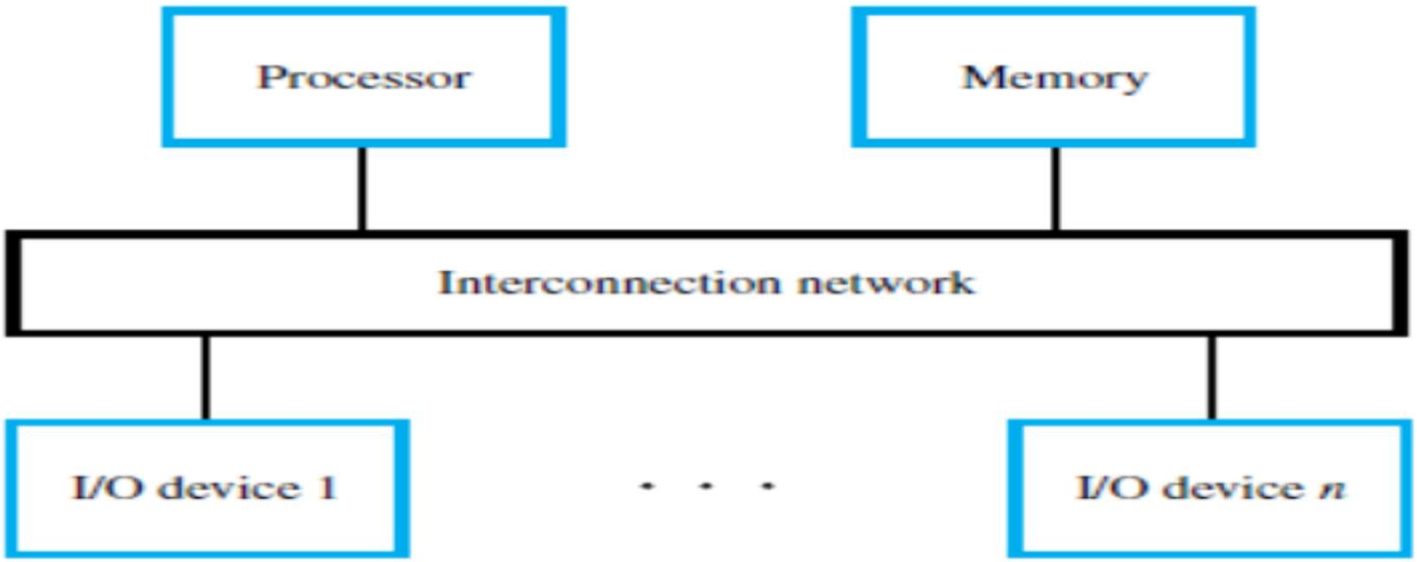


Figure 4.3 Computer System

I/O Module Structure

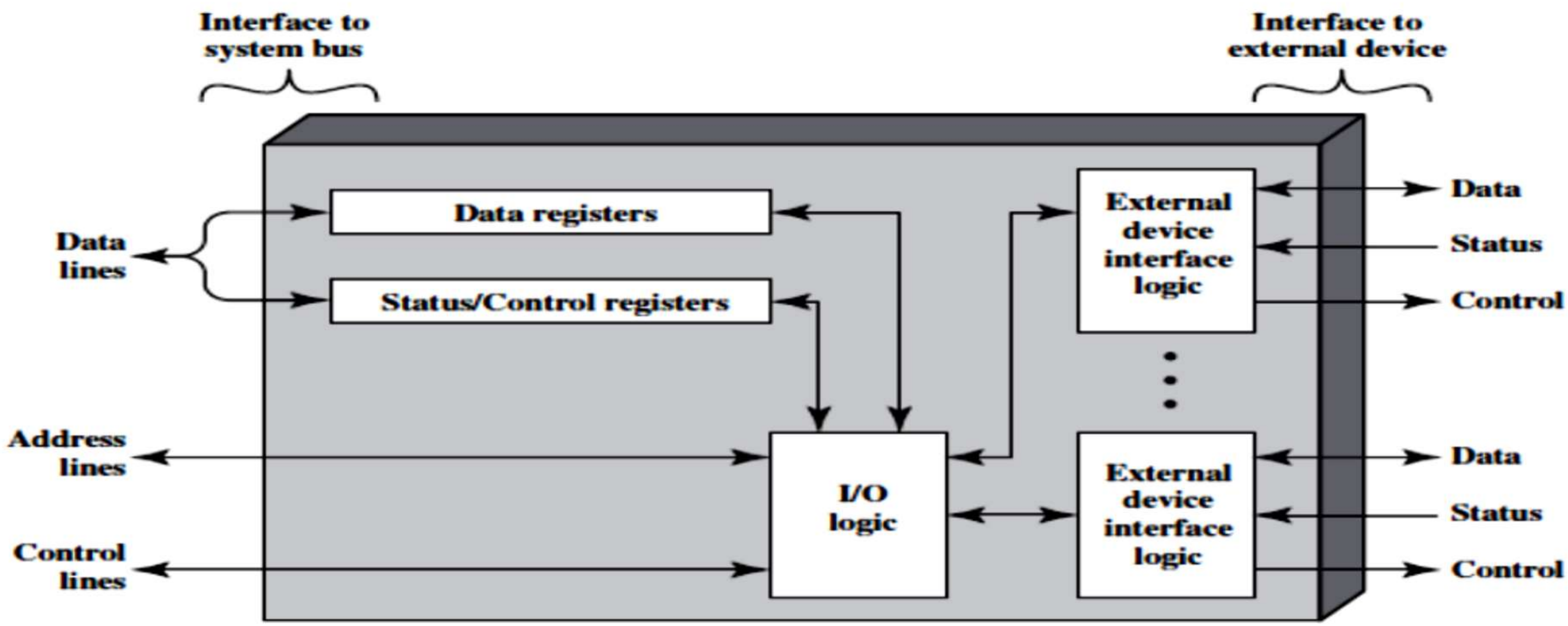


Figure 4.4 Block Diagram of IO Module

I/O Operations

Three techniques are possible for I/O operations:

- **Programmed I/O**, data are exchanged between the processor and the I/O module.
  - 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 waste of processor time.
- **Interrupt-driven I/O**, the processor issues an I/O command, continues to execute other instructions, and is interrupted by the I/O module when the latter has completed its work.
- **Direct memory access (DMA),** In this mode, the I/O module and main memory exchange data directly, without processor involvement.

Programmed I/O

- When the processor is executing a program and encounters an instruction relating to I/O, it executes that instruction by issuing a command to the appropriate I/O module.
- With programmed I/O, the I/O module will perform the requested action and then set the appropriate bits in the I/O status register.
- The I/O module takes no further action to alert the processor.
- In particular, it does not interrupt the processor. Thus, it is the responsibility of the processor to periodically check the status of the I/O module until it finds that the operation is complete.

I/O Commands

To execute an I/O-related instruction, the processor issues an address, specifying the particular I/O module and external device, and an I/O command. There are four types of I/O commands that an I/O module may receive when it is addressed by a processor:

Control: Used to activate a peripheral and tell it what to do.

- For example, a magnetic-tape unit may be instructed to rewind or to move forward one record

Test: Used to test various status conditions associated with an I/O module and its peripherals.

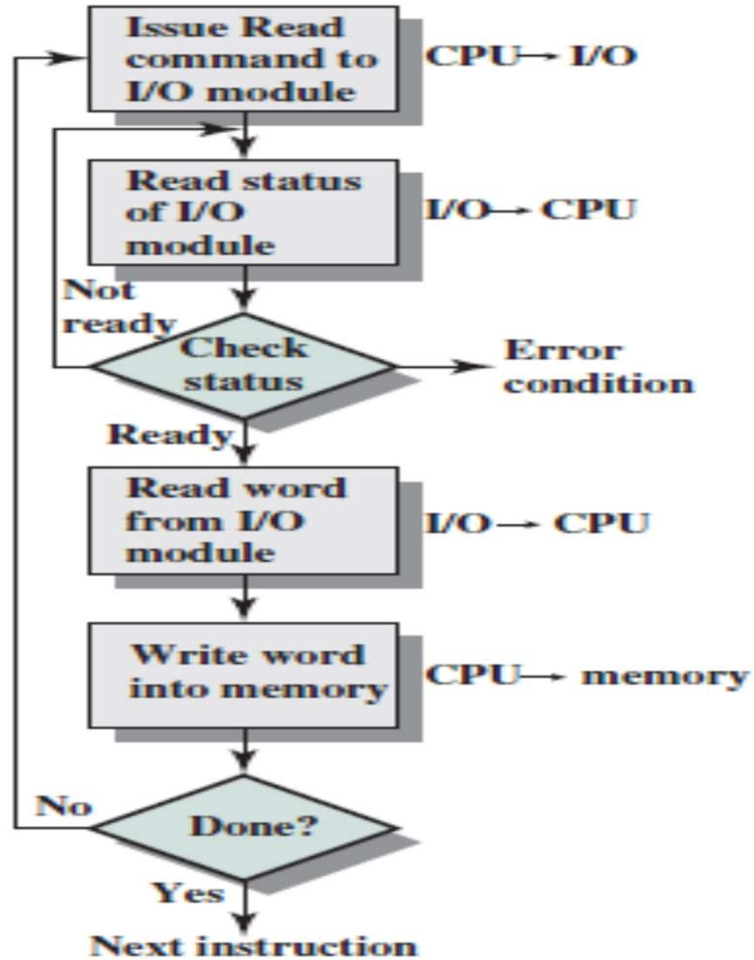
- The processor will want to know that the peripheral of interest is powered on and available for use.
- It will also want to know if the most recent I/O operation is completed and if any errors occurred.

I/O Commands..

Read: Causes the I/O module to obtain an item of data from the peripheral and place it in an internal buffer The processor can then obtain the data item by requesting that the I/O module place it on the data bus.

Write: Causes the I/O module to take an item of data (byte or word) from the data bus and subsequently transmit that data item to the peripheral.

Figure 4.5
Programmed I/O



I/O Instructions

- There will be many I/O devices connected through I/O modules to the system.
- Each device is given a unique identifier or address.
- When the processor issues an I/O command, the command contains the address of the desired device.
- Thus, each I/O module must interpret the address lines to determine if the command is for itself.

I/O Instructions..

- When the processor, main memory, and I/O share a common bus, two modes of addressing are possible: **memory mapped and isolated.**

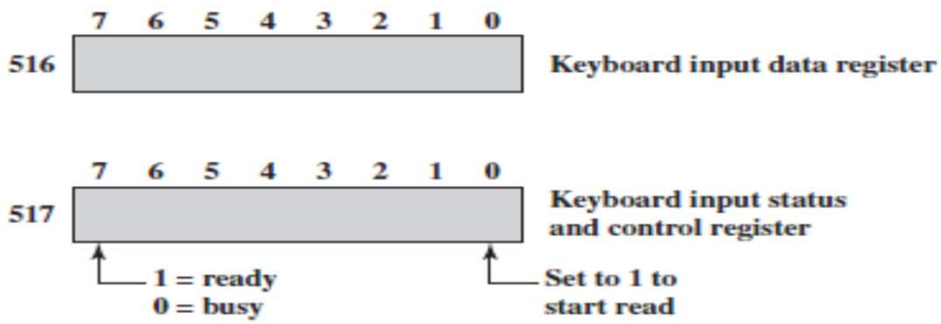
Memory-Mapped I/O

- There is a single address space for memory locations and I/O devices.
- The processor treats the status and data registers of I/O modules as memory locations and uses the same machine instructions to access both memory and I/O devices.
- A single read line and a single write line are needed on the bus

I/O Instructions…

Isolated I/O

- The bus may be equipped with memory read and write plus input and output command lines
- The command line specifies whether the address refers to a memory location or an I/O device.
- The full range of addresses may be available for both
- Because the address space for I/O is isolated from that for memory, this is referred to as **isolated I/O**



ADDRESS	INSTRUCTION	OPERAND	COMMENT
200	Load AC	"1"	Load accumulator
	Store AC	517	Initiate keyboard read
202	Load AC	517	Get status byte
	Branch if Sign = 0	202	Loop until ready
	Load AC	516	Load data byte

(a) Memory-mapped I/O

ADDRESS	INSTRUCTION	OPERAND	COMMENT
200	Load I/O	5	Initiate keyboard read
201	Test I/O	5	Check for completion
	Branch Not Ready	201	Loop until complete
	In	5	Load data byte

(b) Isolated I/O

Figure 4.6 Memory-Mapped and Isolated I/O

Program-Controlled I/O Example

- I/O devices operate at speeds that are very much different from that of the processor
- Keyboard, for example, is very slow
- It needs to make sure that only after a character is available in the input buffer of the keyboard interface; also, this character must be read only once

Program-Controlled I/O Example..

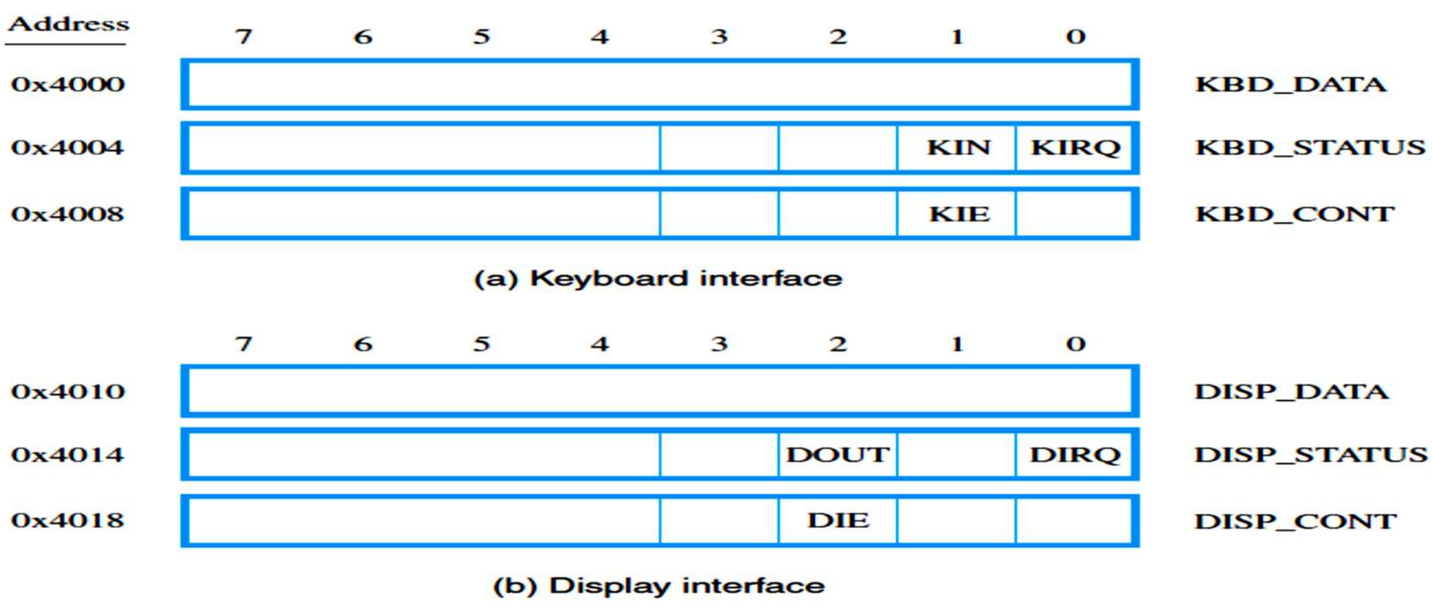


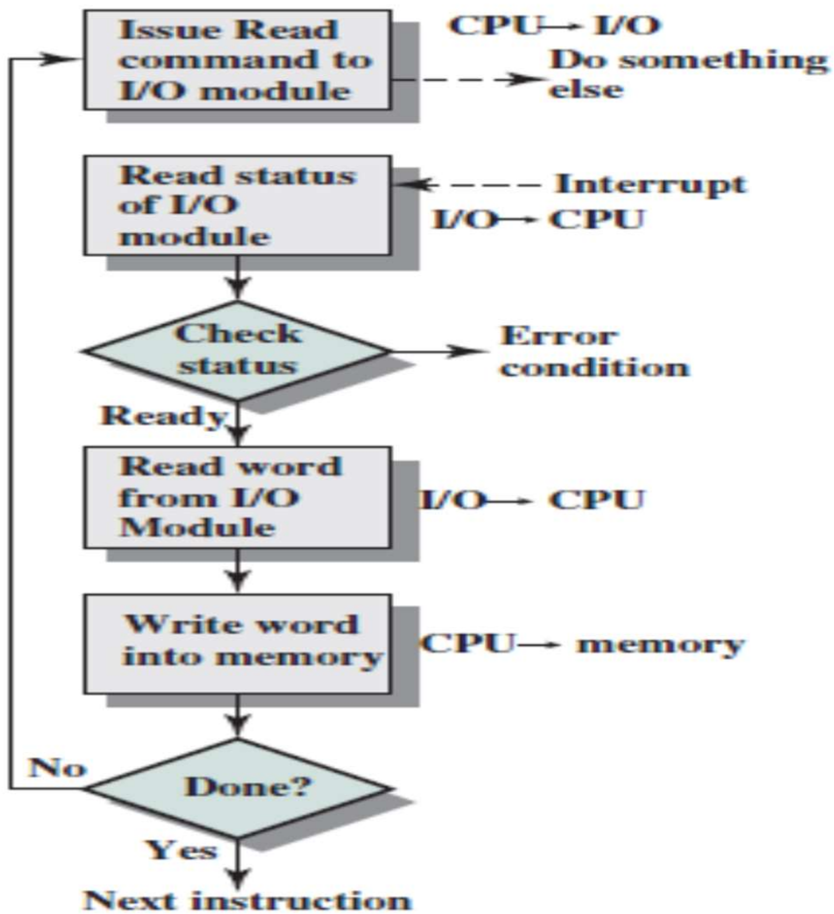
Figure 3.3 Registers in the keyboard and display interfaces.

```
        Move         R2, #LOC          Initialize pointer register R2 to point to the
                                         address of the first location in main memory
                                         where the characters are to be stored.
READ:   TestBit      KBD_STATUS, #1    Wait for a character to be entered
        Branch=0     READ                in the keyboard buffer KBD_DATA.
        MoveByte     (R2), KBD_DATA    Transfer the character from KBD_DATA into
                                         the main memory (this clears KIN to 0).
ECHO:   TestBit      DISP_STATUS, #2   Wait for the display to become ready.
        Branch=0     ECHO
        MoveByte     DISP_DATA, (R2)   Move the character just read to the display
                                         buffer register (this clears DOUT to 0).
        CompareByte  (R2)+, #CR        Check if the character just read is CR
                                         (carriage return). If it is not CR, then
        Branch≠0     READ                branch back and read another character.
                                         Also, increment the pointer to store the
                                         next character.
```

Interrupt-Driven I/O

- The problem with the programmed I/O is the processor enters a wait loop in which it repeatedly tests the device status
- During this period, the processor is not performing any useful computation. There are many situations where other tasks can be performed while waiting for an I/O device to become ready.
- To allow this to happen, we can arrange for the I/O device to alert the processor when it becomes ready.
- It can do so by sending a hardware signal called an interrupt request to the processor. Since the processor is no longer required to continuously poll the status of I/O devices, it can use the waiting period to perform other useful tasks.

Interrupt-Driven I/O..



Interrupt Service Routine

ISR can get invoked from anywhere in the program that was executing –

- This anywhere means it depends on exactly where the interrupt signal arrived.
- So, potentially all the registers that are used in the service routine needs to be saved and restored.

Challenges in Interrupts

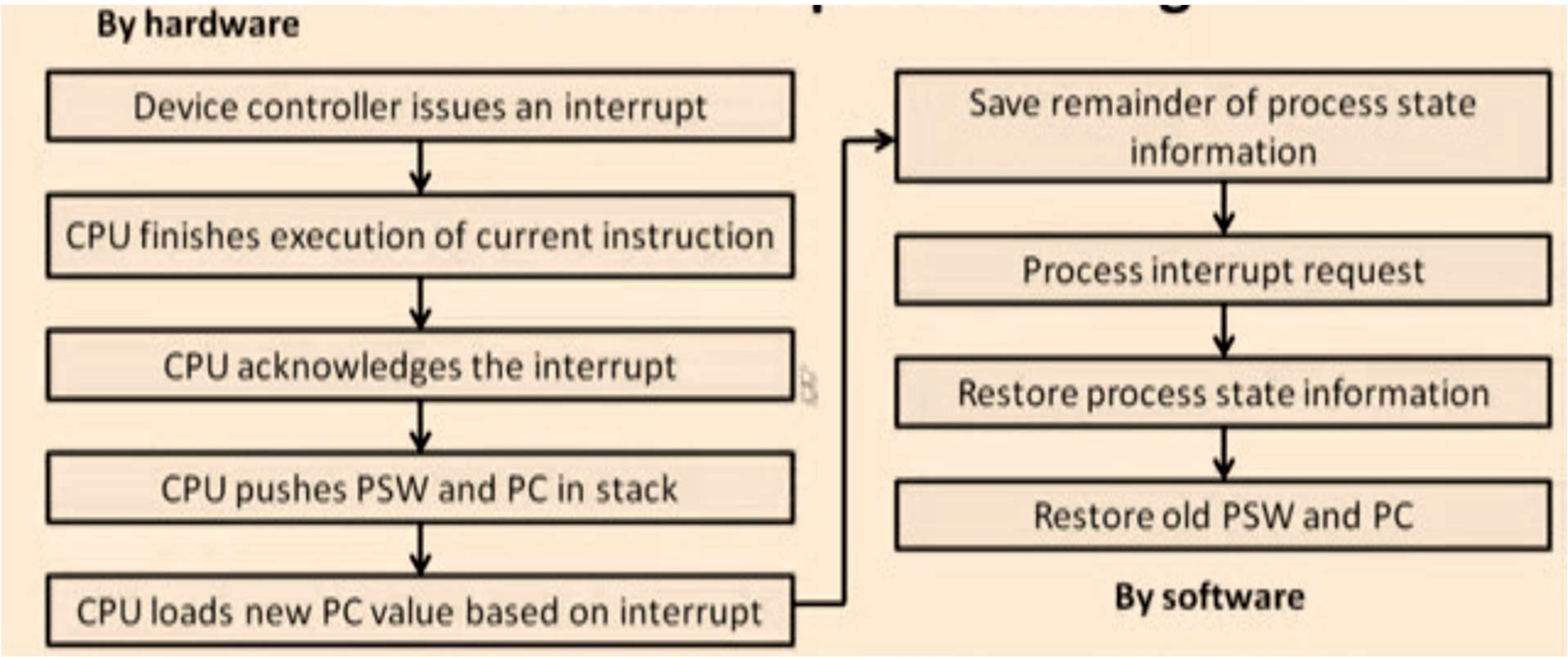
- For multiple sources of the interrupts, how to know the address of the ISR?
- How to handle multiple interrupts?
  - While an interrupt request is being processed, another interrupt request might come.
  - Enabling, disabling and masking of the interrupts
- How to handle simultaneously arriving interrupts?
- Sources of the interrupts other than I/O devices.
  - Exceptions

Interrupt handling

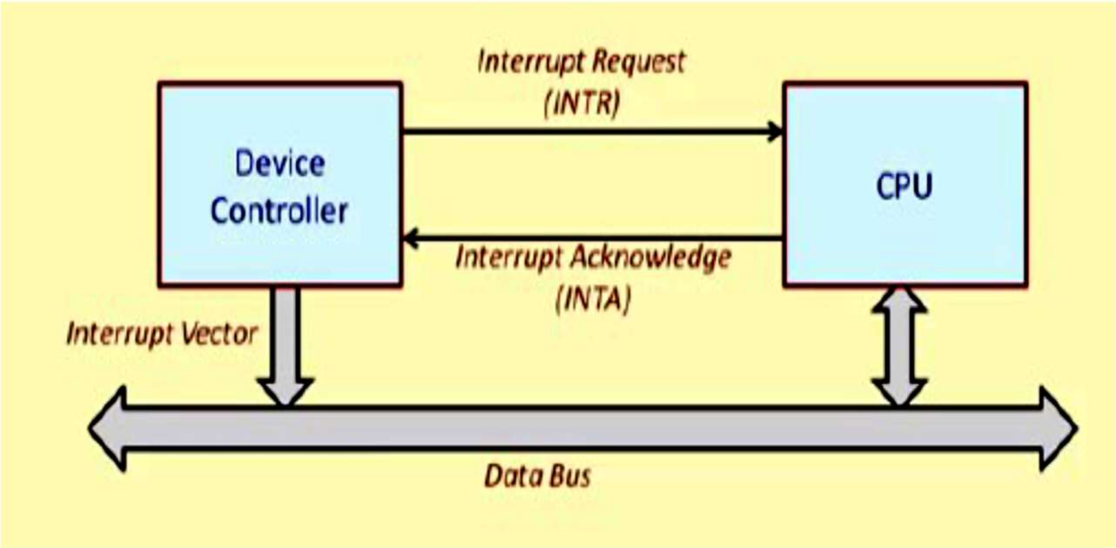
What happens when an interrupt request arrives?

- At the end of the current instruction execution, the PC and program status word(PSW) are saved in the stack automatically.
  - PSW contains status flags and other processor status information.
- The interrupt is acknowledged, the interrupt vector obtained , based on which control transfers to the appropriate ISR.
  - Different interrupting devices may have different ISR'S.
- After handling the interrupt, the ISR executes a special Return From Interrupt (RTI) instruction.
  - Restores the PSW and returns control to the saved PC address
  - Unlike normal RETURN where PSW is not restored .

General Interrupt Processing



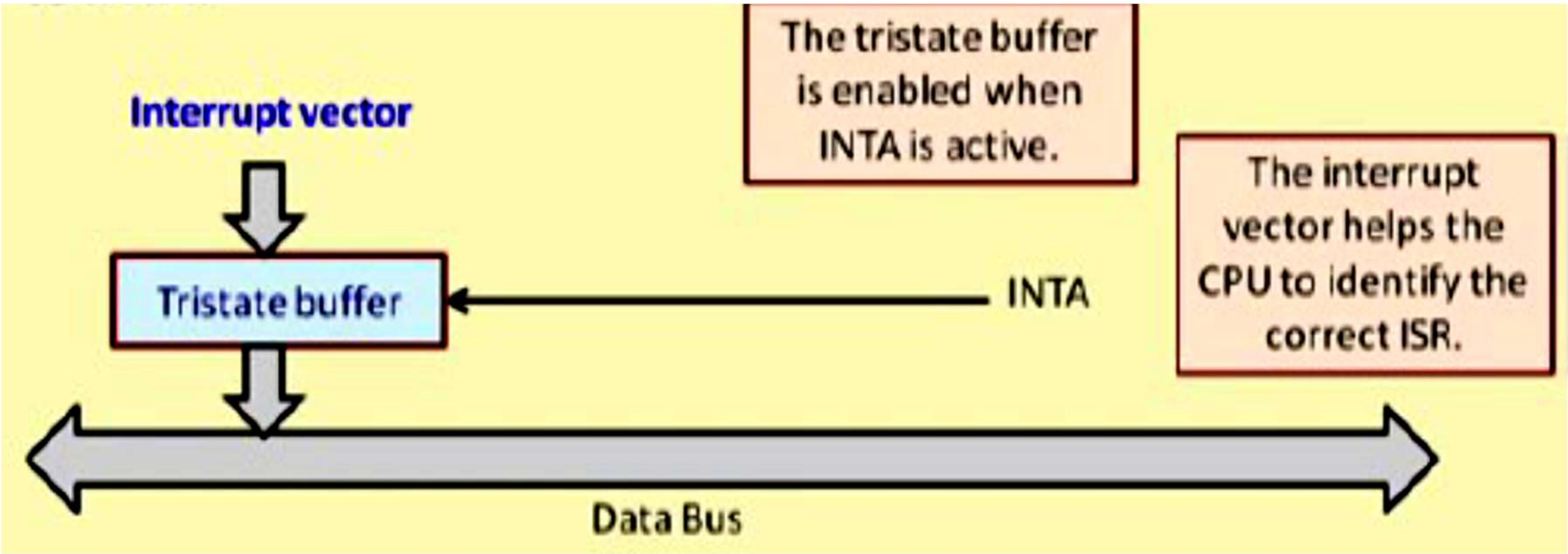
General Interrupt Processing



Steps

- Device controller sends INTR to the CPU.
- CPU finishes the current instruction and sends back INTA.
- Device controller sends interrupt vector over data bus
- CPU reads the interrupt vector, and identifies the device

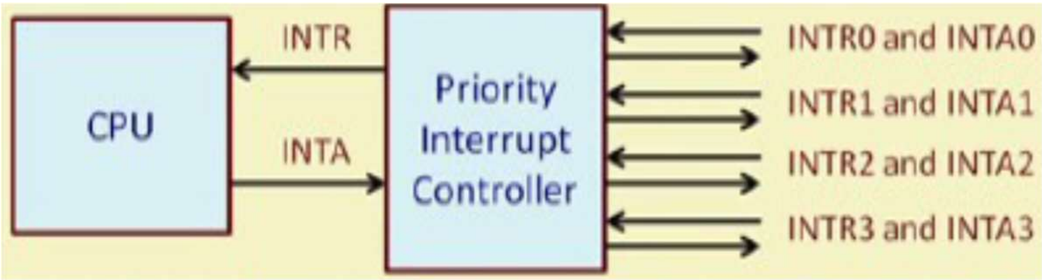
How is the interrupt vector sent on the data bus in response to INTA?



Multiple Devices Interrupting the CPU

A common solution is to use a priority interrupt controller.

- The interrupt controller interacts with CPU on the one side and multiple devices on the other side.
- For simultaneous interrupt requests, interrupt priority is defined.
- The interrupt controller is responsible for sending the interrupt vector to CPU.



How it works?

The INTR line is made active when some of the device(s) activate their interrupt request line.

INTR = INTR0 + INTR1 + INTR2 + INTR3

- When the CPU sends back INTR, the interrupt controller sends back the corresponding acknowledge to the interrupting device, and puts the interrupt vector on the data bus.
- The interrupt controller is programmable, where the interrupt vectors for the various interrupts can be programmed (specified).
- For more than one interrupt request simultaneously active, a priority mechanism is used
  - E.g. INTR0 is highest priority, followed by INTR1, etc.

How is interrupt nesting handled?

Consider the scenario:

- A device D0 has interrupted and the CPU is executing the ISR for D0
- In the mean time, another device D1 has interrupted.

Two possible scenarios here:

- D1 will interrupt the ISR for D0, get processed first, and then the ISP for D0 will be resumed
- Disable the interrupt system automatically whenever an interrupt is acknowledged so that handling of the nested interrupt is not required.

Typical instruction set architectures have the following instructions:

- EI : Enable interrupt
- DI : Disable interrupt

For the second scenario as discussed, the ISR will give an EI instruction just before RTI.

- Some ISA combine EI and RTI in a single instruction.

The DI instruction is sometimes used by the operating system to execute atomic code (e.g. semaphore wait and signal operations).

- Nobody should interrupt the code while it is being executed.

Cases that make interrupt handling difficult

For some interrupts, it is not possible to finish the execution of the current instruction.

- A special RETURN instruction is required that would return and restart the interrupted instructions.

Some examples:

- Page fault interrupt: A memory location is being accessed that is not presently available in main memory.
- Arithmetic exception: Some error has occurred during some arithmetic operations (e.g. division by zero).