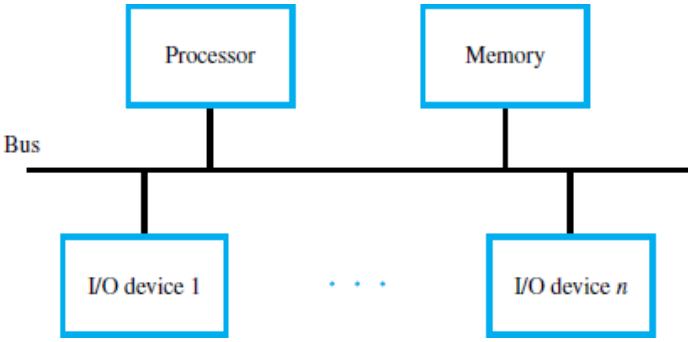
Input-output subsystems

The Input/output organization of computer depends upon the size of computer and the peripherals connected to it. The I/O Subsystem of the computer provides an efficient mode of communication between the central system and the outside environment**.**

The most common input output devices are: Monitor, Keyboard, Mouse, Printer, Magnetic tapes Input Output Interface provides a method for transferring information between internal storage and external I/O devices. Peripherals connected to a computer need special communication links for interfacing them with the central processing unit. The purpose of communication link is to resolve the differences that exist between the central computer and each peripheral.



The Major Differences are:-

- Peripherals are electromechnical and electromagnetic devices and CPU and memory are electronic devices. Therefore, a conversion of signal values may be needed.
- The data transfer rate of peripherals is usually slower than the transfer rate of CPU andconsequently, a synchronization mechanism may be needed.
- Data codes and formats in the peripherals differ from the word format in the CPU and memory.
- The operating modes of peripherals are different from each other and must be controlled so asnot to disturb the operation of other peripherals connected to the CPU.

To resolve these differences, computer systems include special hardware components between the CPU and Peripherals to supervises and synchronizes all input and out transfers. These components are called Interface Units because they interface between the processor bus and the peripheral devices.

I/O device interface

The I/O Bus consists of data lines, address lines and control lines. The I/O bus from the processor is attached to all peripherals interface. To communicate with a particular device, the processor places a device address on address lines. Each Interface decodes the address and control received from the I/O bus, interprets them for peripherals and provides signals for the peripheral controller. It is also synchronizes the data flow and supervises the transfer between peripheral and processor. Each peripheral has its own controller.

For example, the printer controller controls the paper motion, the print timing. The control lines arereferred as I/O command. The commands are as following:
Control command- A control command is issued to activate the peripheral and to inform it what to do. Status command- A status command is used to test various status conditions in the interface and the peripheral.
Data Output command- A data output command causes the interface to respond by transferring datafrom the bus into one of its registers.
Data Input command- The data input command is the opposite of the data output.

In this case the interface receives on item of data from the peripheral and places it in its buffer register. I/O Versus Memory Bus

To communicate with I/O, the processor must communicate with the memory unit. Like the I/O bus, the memory bus contains data, address and read/write control lines. There are 3 ways that computerbuses can be used to communicate with memory and I/O:
1. Use two Separate buses, one for memory and other for I/O.
2. Use one common bus for both memory and I/O but separate control lines for each.
3. Use one common bus for memory and I/O with common control lines.

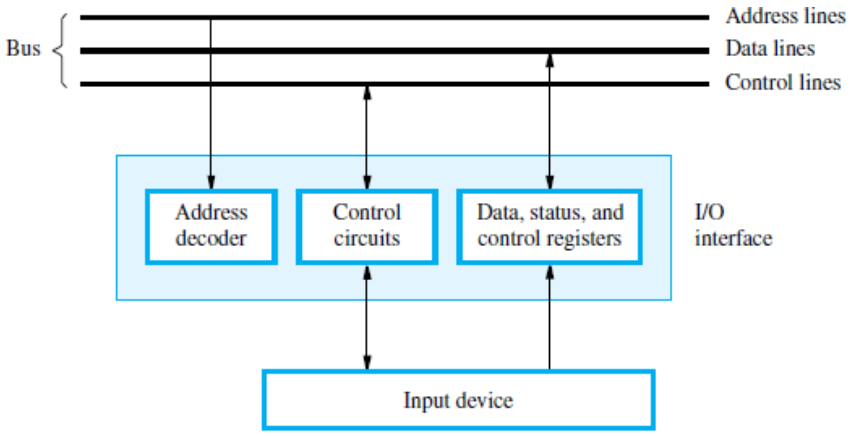


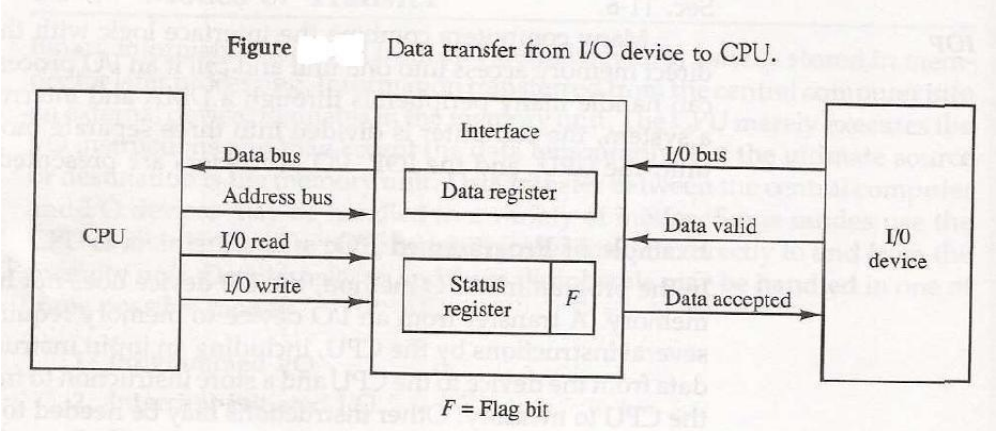
Figure 7.2 I/O interface for an input device.

Programmed I/O Mode:

In this mode of data transfer the operations are the results in I/O instructions which is a part of computer program. Each data transfer is initiated by a instruction in the program. Normally the transfer is from a CPU register to peripheral device or vice- versa. Once the data is initiated the CPU starts monitoring the interface to see when nexttransfer can made. The instructions of the program keep close tabs on everything that takes place in the interface unit and the I/O devices.

The transfer of data requires three instructions:

- Read the status register.
- Check the status of the flag bit and branch to step 1 if not set or to step 3 if set.
- Read the data register.



In this technique CPU is responsible for executing data from the memory for output and storing data in memory for executing of Programmed I/O as shown in Fig.

Drawback of the Programmed I/O:

The main drawback of the Program Initiated I/O was that the CPU has to monitor the units all the times when the program is executing. Thus the CPU stays in a program loop until the I/O unit indicates that it is ready for data transfer. This is a time consuming process and the CPU time is wasted a lot in keeping an eye to the executing of program.

Interrupt-Initiated I/O:

In this method an interrupt facility an interrupt command is used to inform the device about the start and end of transfer. In the meantime the CPU executes other program. When the interface determines that the device is ready for data transfer it generates an Interrupt Request and sends it to the computer.

When the CPU receives such an signal, it temporarily stops the execution of the program and branches to a service program to process the I/O transfer and after completing it returns back to task, what it was originally performing.

In this type of IO, computer does not check the flag. It continues to perform its task. Whenever any device wants the attention, it sends the interrupt signal to the CPU.CPU then deviates from what it was doing, store the return address from PC and branch to the address of the subroutine.

There are two ways of choosing the branch address:

Vectored Interrupt: In vectored interrupt the source that interrupts the CPU provides the branch information. This information is called interrupt vectored. **Non-vectored Interrupt:** In non-vectored interrupt, the branch address is assigned to the fixed address in the memory.

Direct Memory Access (DMA):

In the Direct Memory Access (DMA) the interface transfer the data into and out of the memory unit through the memory bus. The transfer of data between a fast storage device such as magnetic disk and memory is often limited by the speed of the CPU. Removing the CPU from the path and letting the peripheral device manage the memory buses directly would improve the speed of transfer. This transfer technique is called Direct Memory Access (DMA).

During the DMA transfer, the CPU is idle and has no control of the memory buses. A DMA Controller takes over the buses to manage the transfer directly between the I/O device and memory.

The CPU may be placed in an idle state in a variety of ways. One common method extensively used in microprocessor is to disable the buses through special control signals such as:

- Bus Request (BR)
- Bus Grant (BG)

These two control signals in the CPU that facilitates the DMA transfer. The Bus Request (BR) inputis used by the DMA controller to request the CPU. When this input is active, the CPU terminates the execution of the current instruction and places the address bus, data bus and read write lines into a highImpedance state. High Impedance state means that the output is disconnected.

The CPU activates the Bus Grant (BG) output to inform the external DMA that the Bus Request(BR) can now take control of the buses to conduct memory transfer without processor.

When the DMA terminates the transfer, it disables the Bus Request (BR) line. The CPU disables the BusGrant (BG), takes control of the buses and return to its normal operation.

The transfer can be made in several ways

- that are:DMA Burst
- Cycle Stealing

DMA Burst: In DMA Burst transfer, a block sequence consisting of a number of memory words is transferred in continuous burst while the DMA controller is master of the memory buses.

Cycle Stealing: Cycle stealing allows the DMA controller to transfer one data word at a time, afterwhich it must returns control of the buses to the CPU.

DMA Controller:

The DMA controller needs the usual circuits of an interface to communicate with the CPU and I/Odevice. The DMA controller has three registers:

- Address Register
- Word Count Register
- Control Register

Address Register: Address Register contains an address to specify the desired location in memory. **Word Count Register:** WC holds the number of words to be transferred. The register is incre/decre byone after each word transfer and internally tested for zero.
Control Register: Control Register specifies the mode of transfer

The unit communicates with the CPU via the data bus and control lines. The registers in the DMA are selected by the CPU through the address bus by enabling the DS (DMA select) and RS (Register select) inputs. The RD (read) and WR (write) inputs are bidirectional.
When the BG (Bus Grant) input is 0, the CPU can communicate with the DMA registers through the data bus to read from or write to the DMA registers. When BG =1, the DMA can communicate directly with the memory by specifying an address in the address bus and activating the RD or WR control.

DMA Transfer:

The CPU communicates with the DMA through the address and data buses as with any interface unit. The DMA has its own address, which activates the DS and RS lines. The CPU initializes the DMA through the data bus. Once the DMA receives the start control command, it can transfer between the peripheral and the memory.

When BG = 0 the RD and WR are input lines allowing the CPU to communicate with the internal DMA registers. When BG=1, the RD and WR are output lines from the DMA controller to the random access memory to specify the read or write operation of data.

Privileged Instructions and Non- Privileged Instructions:

Instructions are divided into two categories:
- non-privileged instructions
- privileged instructions.

A non-privileged instruction is an instruction that any application or user can execute.

Examples of non-privileged instructions:

```
1 movl
2 addl
3 call
4 ret
```

A privileged instruction, on the other hand, is an instruction that can only be executed in kernel mode. Instructions are divided in this manner because privileged instructions could harm the kernel.

Examples of privileged instructions:

```
insl
outb
inb
int
```

Exceptions and Software interrupts:

Exceptions and interrupts are unexpected events that disrupt the normal flow of instruction execution. An exception is an unexpected event from within the processor. An interrupt is an unexpected event from outside the processor. You are to implement exception and interrupt handling in your multicycle CPU design.

External interrupts come from input input (I/O) devices, from a timing device, from a circuit monitoring the power supply, or from any other external source. Examples that cause external interrupts are l/0 device requesting transfer of data, l/0 device finished transfer of data, elapsed time of an event, or power failure.

Internal interrupts arise from illegal or erroneous use of an instruction or data. Internal interrupts are also called traps. Examples of interrupts caused by internal error conditions are register overflow, attempt to divide by zero, an invalid operation code, stack overflow, and protection violation.

External and internal interrupts are initiated from signals that occur in the hardware of the CPU. A software interrupt is initiated by executing an instruction.

Software interrupt is a special call instruction that behaves like an interrupt rather than a subroutine call. It can be used by the programmer to initiate an interrupt procedure at any desired point in the program.

The most common use of software interrupt is associated with a supervisor call instruction. This instruction provides means for switching from a CPU user mode to the supervisor mode. Certain operations in the computer may be assigned to the supervisor mode only, as for example, a complex input or output transfer procedure.

A program written by a user must run in the user mode. When an input or output transfer is required, the supervisor mode is requested by means of a supervisor call instruction. This instruction causes a software interrupt that stores the old CPU state and brings in a new PSW that belongs to the supervisor mode. The calling program must pass information to the operating system in order to specify the particular task requested.

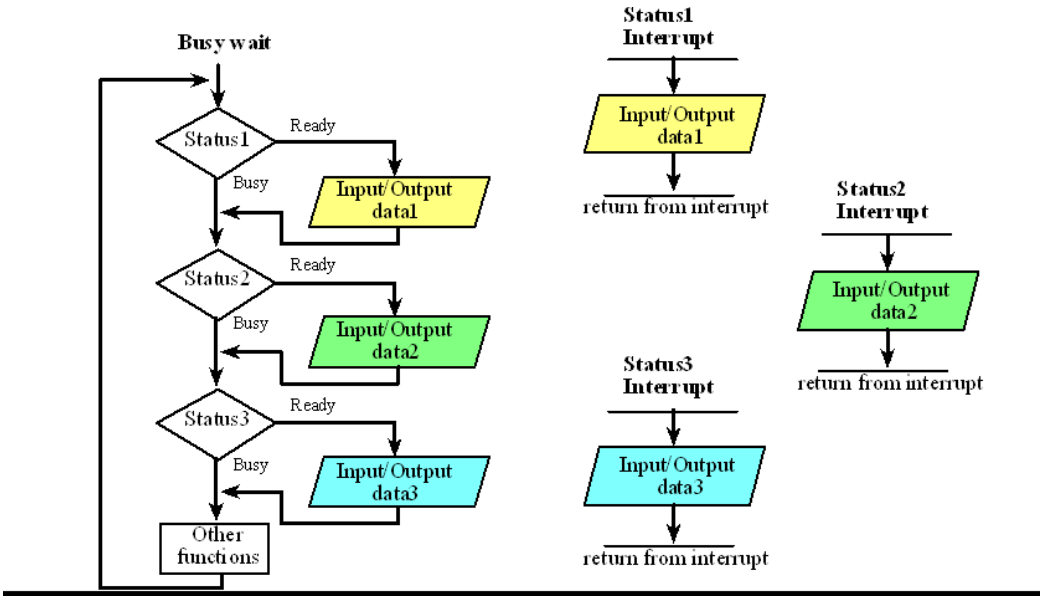
Programs and processes-Role of interrupts in process state transitions

An **interrupt** is the automatic transfer of software execution in response to a hardware event that is asynchronous with the current software execution. This hardware event is called a **trigger**. The hardware event can either be a busy to ready transition in an external I/O device (like the UART input/output) or an internal event (like bus fault, memory fault, or a periodic timer).

When the hardware needs service, signified by a busy to ready state transition, it will request an interrupt by setting its trigger flag. A **thread** is defined as the path of action of software as it executes. The execution of the interrupt service routine is called a background thread. This thread is created by the hardware interrupt request and is killed when the interrupt service routine returns from interrupt (e.g., by executing a **BX LR**). A new thread is created for each interrupt request.

It is important to consider each individual request as a separate thread because local variables and registers used in the interrupt service routine are unique and separate from one interrupt event to the next interrupt. In a **multi-threaded** system, we consider the threads as cooperating to perform an overall task. Consequently we will develop ways for the threads to communicate (e.g., FIFO) and to synchronize with each other. Most embedded systems have a single common overall goal.

On the other hand, general-purpose computers can have multiple unrelated functions to perform. A **process** is also defined as the action of software as it executes. Processes do not necessarily cooperate towards a common shared goal. Threads share access to I/O devices, system resources, and global variables, while processes have separate global variables and system resources. Processes do not share I/O devices.



I/O Device Interfaces

SCSI:

The acronym SCSI stands for Small Computer System Interface. It refers to a standard bus
defined by the American National Standards Institute (ANSI) under the designation X3.131 . In the original specifications of the standard, devices such as disks are connected to a computer via a 50-wirecable, which can be up to 25 meters in length and can transfer data at rates up to 5 megabytes/s. The SCSI bus standard has undergone many revisions, and its data transfer capability has increased very rapidly, almost doubling every two years. SCSI-2 and SCSI-3 have been defined, and each has several options.

A SCSI bus may have eight data lines, in which case it is called a narrow bus and transfers data one byte at a time. Alternatively, a wide SCSI bus has 16 data lines and transfers data 16 bits at a time. There are also several options for the electrical signaling scheme used. Devices connected to the SCSI bus are not part of the address space of the processor in the same way as devices connected to the processor bus. The SCSI bus is connected to the processor bus through a SCSI controller. This controller uses DMA to transfer data packets from the main memory to the device, or vice versa. A packet may contain a block of data, commands from the processor to the device, or status information about the device.

To illustrate the operation of the SCSI bus, let us consider how it may be used with a disk drive. Communication with a disk drive differs substantially from communication with the main memory. A controller connected to a SCSI bus is one of two types – an initiator or a target. An initiator has the ability to select a particular target and to send commands specifying the operations to be performed.

Clearly, the controller on the processor side, such as the SCSI controller, must be able to operate as an initiator. The disk controller operates as a target. It carries out the

commands it receives from the initiator. The initiator establishes a logical connection with the intended target. Once this connection has been established, it can be suspended and restored as needed to transfer commands and bursts of data.

While a particular connection is suspended, other device can use the bus to transfer information. This ability to overlap data transfer requests is one of the key features of the SCSI bus that leads to its high performance.

Data transfers on the SCSI bus are always controlled by the target controller. To send a command to a target, an initiator requests control of the bus and, after winning

---

arbitration, selects the controller it wants to communicate with and hands control of the bus over to it.

Then the controller starts a data transfer operation to receive a command from the initiator.

The processor sends a command to the SCSI controller, which causes the following sequence ofevent to take place:

1. The SCSI controller, acting as an initiator, contends for control of the bus.
2. When the initiator wins the arbitration process, it selects the target controller and hands over controlof the bus to it.
3. The target starts an output operation (from initiator to target); in response to this, the initiator sends acommand specifying the required read operation.
4. The target, realizing that it first needs to perform a disk seek operation, sends a message to the initiator indicating that it will temporarily suspend the connection between them. Then it releases thebus.
5. The target controller sends a command to the disk drive to move the read head to the first sector involved in the requested read operation. Then, it reads the data stored in that sector and stores them in a data buffer. When it is ready to begin transferring data to the initiator, the target requests control of the bus. After it wins arbitration, it reselects the initiator controller, thus restoring the suspended connection.
6. The target transfers the contents of the data buffer to the initiator and then suspends the connection again. Data are transferred either 8 or 16 bits in parallel, depending on the width of the bus.
7. The target controller sends a command to the disk drive to perform another seek operation. Then, it transfers the contents of the second disk sector to the initiator as before. At the end of these transfers, the logical connection between the two controllers is terminated.
8. As the initiator controller receives the data, it stores them into the main memory using the DMA approach.
9. The SCSI controller sends as interrupt to the processor to inform it that the requested operation hasbeen completed.

This scenario show that the messages exchanged over the SCSI bus are at a higher level than those exchanged over the processor bus. In this context, a “higher level” means that the messages refer to operations that may require several steps to complete, depending on the device. Neither the processor nor the SCSI controller need be aware of the details of operation of the particular device involved in a data transfer. In the preceding example, the processor need not be involved in the disk seek operation.

USB

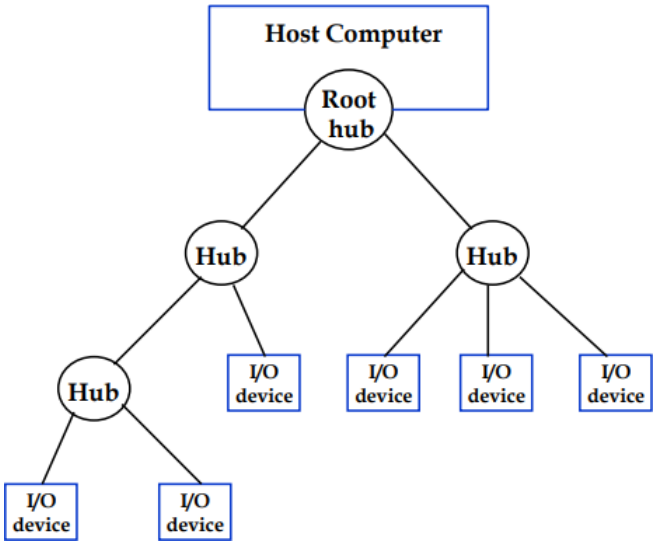
The USB has been designed to meet several key objectives:

- Provide a simple, low-cost, and easy to use interconnection system that overcomes the difficulties due to the limited number of I/O ports available on a computer. ‹
- Accommodate a wide range of data transfer characteristics for I/O devices, including telephone and Internet connections. ‹
- Enhance user convenience through a "plug-and-play" mode of operation.

USB Structure

- A serial transmission format has been chosen for the USB because a serial bus satisfies the low-cost and flexibility requirements.
- Clock and data information are encoded together and transmitted as a single signal. Hence, there are no limitations on clock frequency or distance arising from data skew.
- To accommodate a large number of devices that can be added or removed at any time, the USB has the tree structure. Each node of the tree has a device called a hub, which acts as an intermediate control point between the host and the I/O device. At the root of the tree, a root hub connects the entire tree to the host computer.
- The tree structure enables many devices to be connected while using only simple point-to-point serial links.
- Each hub has a number of ports where devices may be connected, including other hubs.

- In normal operation, a hub copies a message that it receives from its upstream connection to all its downstream ports. As a result, a message sent by the host computer is broadcast to all I/O devices, but only the addressed device will respond to that message.
- A message sent from an I/O device is sent only upstream towards the root of the tree and is not seen by other devices. Hence, USB enables the host to communicate with the I/O devices, but it does not enable these devices to communicate with each other.



USB Protocols:

- All information transferred over the USB is organized in packets, where a packet consists of one or more bytes of information.
- The information transferred on the USB can be divided into two broad categories: control and data. ‹ Control packets perform such tasks as addressing a device to initiate data transfer, acknowledging that data have been received correctly, or indicating an error. Data packets carry information that is delivered to a device. For example, input and output data are transferred inside data packets