Computer Organization and Architecture
Module-4

Input/Output Organization: Accessing I/O Devices, Interrupts – Interrupt Hardware, Enabling and Disabling Interrupts, Handling Multiple Devices, Controlling Device Requests, Exceptions, Direct Memory Access, Buses, Interface Circuits, Standard I/O Interfaces – PCI Bus, SCSI Bus, USB.

Q Draw the arrangement of a single bus structure and brief about memory mapped I/O. **Jan 2014**

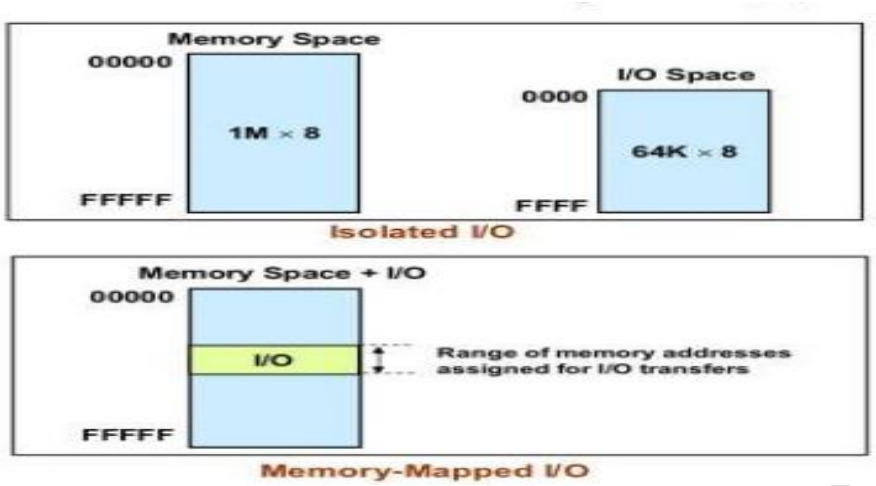
ACCESSING I/O-DEVICES

The components of a computer system communicate with each other through an interconnection network, as shown in Figure 3.1. The interconnection network consists of circuits needed to transfer information between the processor, the memory unit, and a number of I/O devices.

• A **single bus-structure** can be used for connecting I/O-devices to a computer (Figure 7.1).
• Each I/O device is assigned a unique set of address.
• Bus consists of 3 sets of lines to carry address, data & control signals.
• When processor places an address on address-lines, the intended-device responds to the command.
• The processor requests either a read or write-operation.
• The requested-data are transferred over the data-lines.

Figure 3.1 Single bus structure

• There are 2 ways to deal with I/O-devices:
1) Memory-mapped I/O &
2) I/O-mapped I/O/(Isolated I/O)



1) Memory-Mapped I/O

Memory and I/O-devices share a common address-space.
Any data-transfer instruction (like Move, Load) can be used to exchange information.

For example,

Move DATAIN, R0; This instruction sends the contents of location DATAIN to register R0.

Here, DATAIN address of the input-buffer of the keyboard.

2) I/O-Mapped I/O(Isolated I/O)

Memory and I/0 address-spaces are different.
A special instructions named **IN** and **OUT** are used for data-transfer.
Advantage of separate I/O space: I/O-devices deal with fewer address-lines.

IN 19H, R0 ; This instruction stores the data in R0 from the I/O port 19H

Here, 19H refers to I/O Port.

Comparison - Memory-mapped vs port-mapped

Memory-mapped IO	Port-mapped IO
Same address bus to address memory and I/O devices	Different address spaces for memory and I/O devices
Access to the I/O devices using regular instructions	Uses a special class of CPU instructions to access I/O devices
Most widely used I/O method	x86 Intel microprocessors - IN and OUT instructions

I/O Interface for an Input Device

An I/O device is connected to the interconnection network by using a circuit, called the *device interface*, which provides the means for data transfer and for the exchange of status and control information needed to facilitate the data transfers and govern the operation of the device. The interface includes some registers that can be accessed by the processor. One register may serve as a buffer for data transfers, another may hold information about the current status of the device, and yet another may store the information that controls the operational behavior of the device.

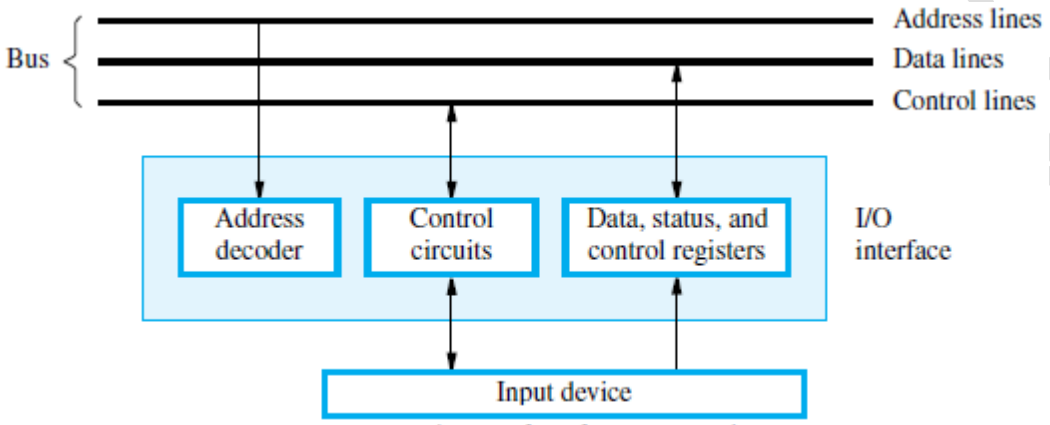


Figure 7.2 I/O interface for an input device.

1) Address Decoder: enables the device to recognize its address when this address appears on the address-lines (Figure 7.2).

2) Status Register: contains information relevant to operation of I/O-device.

3) Data Register: holds data being transferred to or from processor. There are 2 types:

i) DATAIN Input-buffer associated with keyboard.
ii) DATAOUT Output data buffer of a display/printer.

MECHANISMS USED FOR INTERFACING I/O-DEVICES

1) Program Controlled I/O

Consider a task that reads characters typed on a keyboard, stores these data in the memory, and displays the same characters on a display screen. A simple way of implementing this task is to write a program that performs all functions needed to realize the desired action. This method is known as *program-controlled I/O*.

• Processor repeatedly checks status-flag to achieve required synchronization b/w processor & I/O device. (We say that the processor polls the device).
• Main drawback:
The processor wastes time in checking status of device before actual data-transfer takes place.

2) Interrupt I/O

• I/O-device initiates the action instead of the processor.
• I/O-device sends an INTR signal over bus whenever it is ready for a data-transfer operation.
• Like this, required synchronization is done between processor & I/O device.

3) Direct Memory Access (DMA)
• Device-interface transfer data directly to/from the memory w/o continuous involvement by the processor.
• DMA is a technique used for high speed I/O-device.

Q Define and explain briefly the following
i) interrupt

INTERRUPTS
• There are many situations where other tasks can be performed while waiting for an I/O device to become ready.
• A hardware signal called an Interrupt will alert the processor when an I/O device becomes ready.
• Interrupt-signal is sent on the interrupt-request line.
• The processor can be performing its own task without the need to continuously check the I/O-device.
• The routine executed in response to an interrupt-request is called ISR.
• The processor must inform the device that its request has been recognized by sending INTA signal.

(INTR :Interrupt Request, INTA :Interrupt Acknowledge, ISR :Interrupt Service Routine)

• For example, consider COMPUTE and PRINT routines (Figure 3.6).

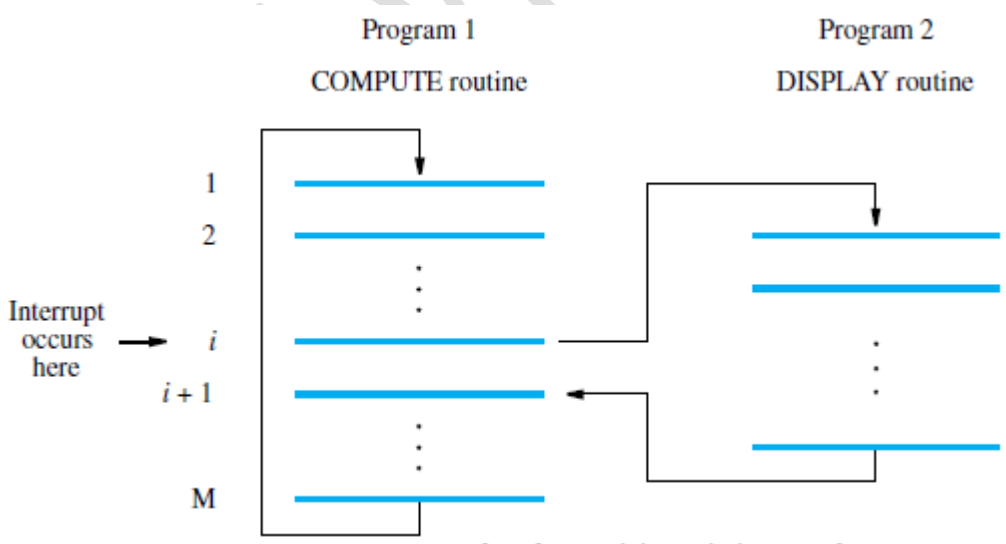


Figure 3.6 Transfer of control through the use of interrupts.

• The processor first completes the execution of instruction i.
• Then, processor loads the PC with the address of the first instruction of the ISR.
• After the execution of ISR, the processor has to come back to instruction i+1.

• Therefore, when an interrupt occurs, the current content of PC is put in temporary storage location.
• A return at the end of ISR reloads the PC from that temporary storage location.
• This causes the execution to resume at instruction i+1.
• When processor is handling interrupts, it must inform device that its request has been recognized.
• This may be accomplished by INTA signal.
• The task of saving and restoring the information can be done automatically by the processor.
• The processor saves only the contents of **PC & Status register.**
• Saving registers also increases the Interrupt Latency.

• **Interrupt Latency** is a delay between

→ time an interrupt-request is received and
→ start of the execution of the ISR.
• Generally, the long interrupt latency in unacceptable.

Difference between Subroutine & ISR

Subroutine	ISR
A subroutine performs a function required by the program from which it is called.	ISR may not have anything in common with program being executed at time INTR is received
Subroutine is just a linkage of 2 or more function related to each other.	Interrupt is a mechanism for coordinating I/O transfers.

INTERRUPT HARDWARE(Single interrupt-request (IR) line)

• Most computers have several I/O devices that can request an interrupt.
• A single interrupt-request (IR) line may be used to serve n devices (Figure 4.6).
• All devices are connected to IR line via switches to ground.
• To request an interrupt, a device closes its associated switch.
• Thus, if all IR signals are inactive, the voltage on the IR line will be equal to Vdd.
• When a device requests an interrupt, the voltage on the line drops to 0.
• This causes the INTR received by the processor to go to 1.
• The value of INTR is the logical OR of the requests from individual devices.
INTR=INTR1+ INTR2+ . . . . . +INTRn

• A special gates known as open-collector or open-drain are used to drive the INTR line.
• The Output of the open collector control is equal to a switch to the ground that is
→ open when gates input is in ”0“ state and
→ closed when the gates input is in “1“ state.
• Resistor R is called a **Pull-up Resistor** because
it pulls the line voltage up to the high-voltage state when the switches are open.

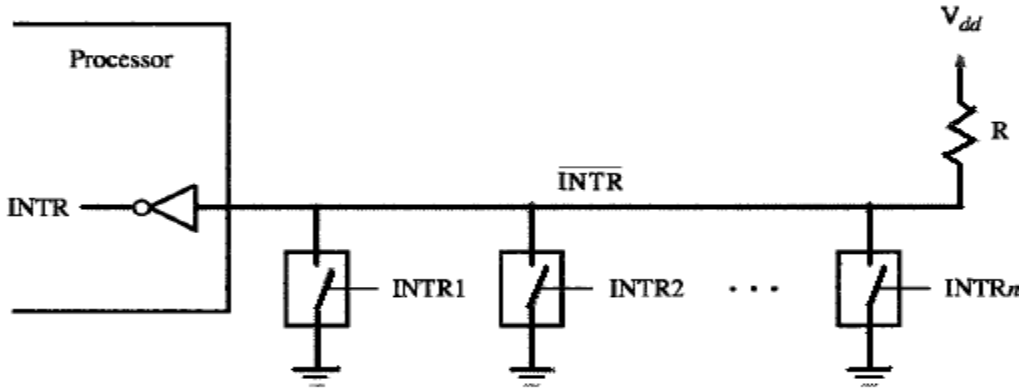


Figure 4.6 An equivalent circuit for an open-drain bus used to implement a common interrupt-request line.

Q Discuss the different schemes available to disable and enable the interrupts **June 2014 ,July 2016**
Q Discuss the different schemes available to disable and enable the interrupts(8M)**Dec 2015**

ENABLING & DISABLING INTERRUPTS

The facilities provided in a computer must give the programmer complete control over the events that take place during program execution. The arrival of an interrupt request from an external device causes the processor to suspend the execution of one program and start the execution of another. Because interrupts can arrive at any time, they may alter the sequence of events from that envisaged by the programmer. Hence, the interruption of program execution must be carefully controlled. A fundamental facility found in all computers is the ability to enable and disable such interruptions as desired.

• All computers fundamentally should be able to enable and disable interruptions as desired.
• The problem of infinite loop occurs due to successive interruptions of active INTR signals.
• There are 3 mechanisms to solve problem of infinite loop:

1) Processor should ignore the interrupts until execution of first instruction of the ISR.
2) Processor should automatically disable interrupts before starting the execution of the ISR.
3) Processor has a special INTR line for which the interrupt-handling circuit. Interrupt-circuit responds only to leading edge of signal. Such line is called edge-triggered.

• Sequence of events involved in handling an interrupt-request:
1) The device raises an interrupt-request.
2) The processor interrupts the program currently being executed.
3) Interrupts are disabled by changing the control bits in the processor status register (PS).
4) The device is informed that its request has been recognized. In response, the device deactivates the interrupt-request signal.
5) The action requested by the interrupt is performed by the interrupt-service routine.
6) Interrupts are enabled and execution of the interrupted program is resumed.

HANDLING MULTIPLE DEVICES

While handling multiple devices, the issues concerned are:

1) How can the processor recognize the device requesting an interrupt?
2) How can the processor obtain the starting address of the appropriate ISR?
3) Should a device be allowed to interrupt the processor while another interrupt is being serviced?
4) How should 2 or more simultaneous interrupt-requests be handled?

1)POLLING

When a request is received over the common interrupt-request line, additional information is needed to identify the particular device that activated the line. The information needed to determine whether a device is requesting an interrupt is available in its status register. When a device raises an interrupt request, it sets to 1 one of the bits in its status register, which we will call the IRQ bit. For example, bits KIRQ and DIRQ are the interrupt request bits for the keyboard and the display, respectively. The simplest way to identify the interrupting device is to have the interrupt-service routine poll all the I/O devices connected to the bus. The first device encountered with its IRQ bit set is the device that should be serviced. An appropriate subroutine is called to provide the requested service.

DIRQ Interrupt-request for display.
KIRQ Interrupt-request for keyboard.
KEN keyboard enable.
DEN Display Enable.
SIN, SOUT status flags.

The polling scheme is easy to implement. Its main disadvantage is the time spent interrogating the IRQ bits of all the devices that may not be requesting any service. An alternative approach is to use vectored interrupts, which we describe next.

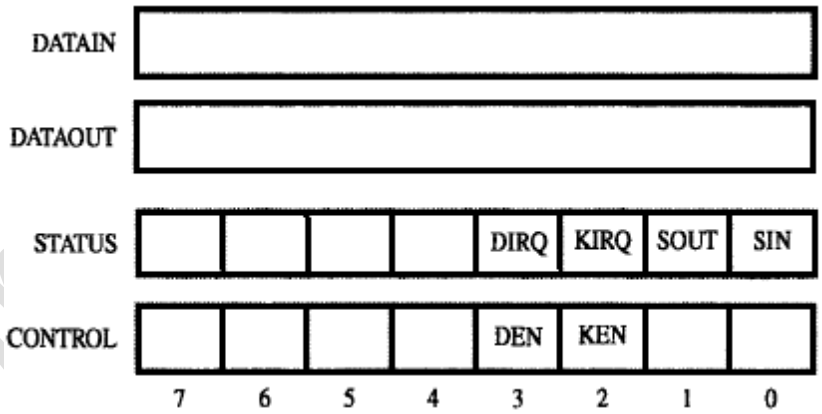


Figure 4.3 Registers in keyboard and display interfaces.

```
        Move        #LINE,R0          Initialize memory pointer.
WAITK   TestBit     #0,STATUS         Test SIN.
        Branch=0    WAITK             Wait for character to be entered.
        Move        DATAIN,R1         Read character.
WAITD   TestBit     #1,STATUS         Test SOUT.
        Branch=0    WAITD             Wait for display to become ready.
        Move        R1,DATAOUT        Send character to display.
        Move        R1,(R0)+          Store charater and advance pointer.
        Compare     #$0D,R1           Check if Carriage Return.
        Branch≠0    WAITK             If not, get another character.
        Move        #$0A,DATAOUT      Otherwise, send Line Feed.
        Call        PROCESS           Call a subroutine to process the
                                        the input line.
```

Figure 4.4 A program that reads one line from the keyboard, stores it in memory buffer, and echoes it back to the display.

Q Define and explain briefly the following
i)vectored interrupt

2) VECTORED INTERRUPTS

• A device requesting an interrupt identifies itself by sending a special-code to processor over bus.
• Then, the processor starts executing the ISR.
• The special-code indicates starting-address of ISR.
• The special-code length ranges from 4 to 8 bits.
• The location pointed to by the interrupting-device is used to store the staring address to ISR.
• The staring address to ISR is called the **interrupt vector**.
• Processor
→ loads interrupt-vector into PC &
→ executes appropriate ISR.
• When processor is ready to receive interrupt-vector code, it activates INTA line.
• Then, I/O-device responds by sending its interrupt-vector code & turning off the INTR signal.
• The interrupt vector also includes a new value for the Processor Status Register.

CONTROLLING DEVICE REQUESTS

• Following condition-codes are used:
KEN Keyboard Interrupt Enable.
DEN Display Interrupt Enable.
KIRQ/DIRQ Keyboard/Display unit requesting an interrupt.

Q . Define and explain briefly the following
i) Interrupt nesting

3)INTERRUPT NESTING

• A multiple-priority scheme is implemented by using separate INTR & INTA lines for each device
• Each INTR line is assigned a different priority-level (Figure 4.7).
• Priority-level of processor is the priority of program that is currently being executed.

• Processor accepts interrupts only from devices that have higher-priority than its own.
• At the time of execution of ISR for some device, priority of processor is raised to that of the device.
• Thus, interrupts from devices at the same level of priority or lower are disabled.

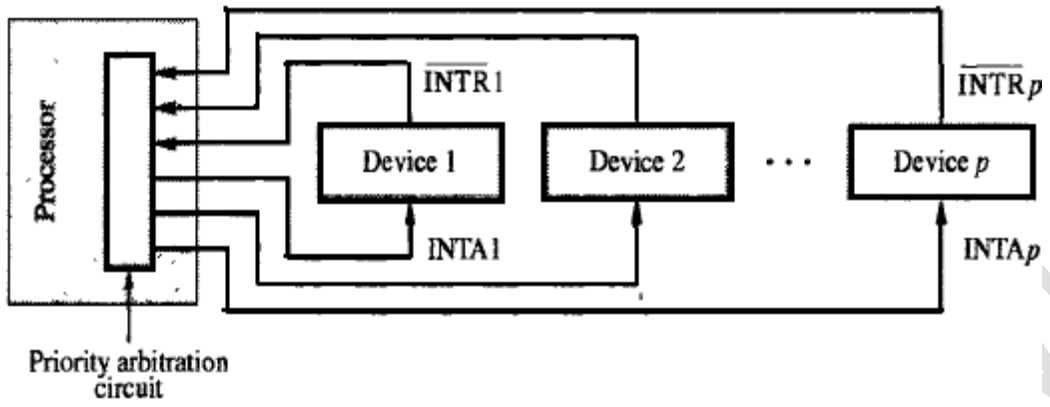


Figure 4.7 Implementation of interrupt priority using individual interrupt-request and acknowledge lines.

This example suggests that I/O devices should be organized in a priority structure. An interrupt request from a high-priority device should be accepted while the processor is servicing another request from a lower-priority device. A multiple-level priority organization means that during execution of an interrupt-service routine, interrupt requests will be accepted from some devices but not from others, depending upon the device's priority. To implement this scheme, we can assign a priority level to the processor that can be changed under program control.

The priority level of the processor is the priority of the program that is currently being executed. The processor accepts interrupts only from devices that have priorities higher than its own. The processor's priority is usually encoded in a few bits of the processor status word. It can be changed by program instructions that write into the PS.

These are privileged instructions, which can be executed only while the processor is running in the supervisor mode. The processor is in the supervisor mode only when executing operating system routines. It switches to the user mode before beginning to execute application programs.

Thus, a user program cannot accidentally, or intentionally, change the priority of the processor and disrupt the system's operation. An attempt to execute a privileged instruction while in the user mode leads to a special type of interrupt called a privileged instruction.

A multiple-priority scheme can be implemented easily by using separate interrupt-request and interrupt-acknowledge lines for each device, as shown in figure. Each of the interrupt-request lines is assigned a different priority level. Interrupt requests received over these lines are sent to a priority arbitration circuit in the processor. A request is accepted only if it has a higher priority level than that currently assigned to the processor.

Q.Explain in with the help of a diagram the working of daisy chain with multiple priority levels and multiple devices in each level .**Dec 2014,Jan 2015**

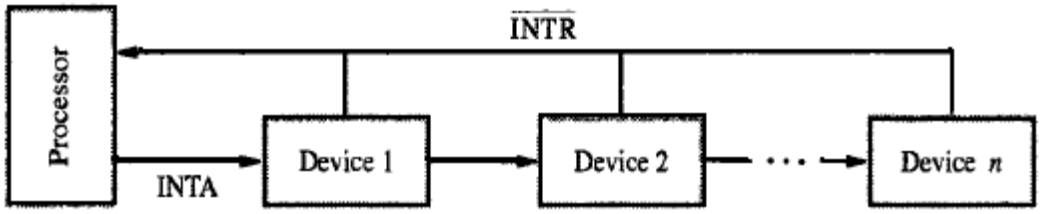
4) SIMULTANEOUS REQUESTS

• The processor must have some mechanisms to decide which request to service when simultaneous requests arrive.
• INTR line is common to all devices (Figure 4.8a).
• INTA line is connected in a daisy-chain fashion.
• INTA signal propagates serially through devices.
• When several devices raise an interrupt-request, INTR line is activated.
• Processor responds by setting INTA line to 1. This signal is received by device 1.
• Device-1 passes signal on to device 2 only if it does not require any service.
• If device-1 has a pending-request for interrupt, the device-1
→ blocks INTA signal &
→ proceeds to put its identifying-code on data-lines.
• Device that is electrically closest to processor has highest priority.

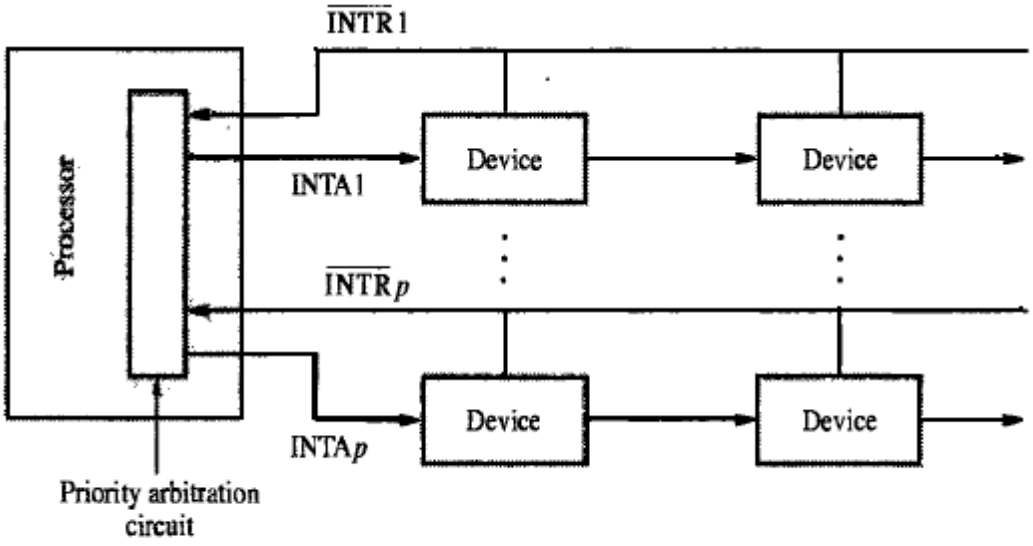
• **Advantage:** It requires fewer wires than the individual connections.

Arrangement of Priority Groups

• Here, the devices are organized in groups & each group is connected at a different priority level.
• Within a group, devices are connected in a daisy chain. (Figure 4.8b).



(a) Daisy chain



(b) Arrangement of priority groups

Figure 4.8 Interrupt priority schemes.

Q Define and explain briefly the following
i)an exception and give two examples

EXCEPTIONS

An interrupt is an event that causes the execution of one program to be suspended and the execution of another program to begin. So far, we have dealt only with interrupts caused by events associated with I/O data transfers. However, the interrupt mechanism is used in a number of other situations.

The term *exception* is often used to refer to any event that causes an interruption. Hence, I/O interrupts are one example of an exception.

• An **interrupt** is an event that causes
→ execution of one program to be suspended &
→ execution of another program to begin.

• **Exception** refers to any event that causes an interruption. For ex: I/O interrupts.

1. Recovery from Errors
• These are techniques to ensure that all hardware components are operating properly.
• For ex: Many computers include an ECC in memory which allows detection of errors in stored-data.
(ECC : Error Checking Code, ESR : Exception Service Routine).
• If an error occurs, control-hardware
→ detects the errors &
→ informs processor by raising an interrupt.

• When exception processing is initiated (as a result of errors), processor.
→ suspends program being executed &
→ starts an ESR. This routine takes appropriate action to recover from the error.

2. Debugging
• Debugger
→ is used to find errors in a program and
→ uses exceptions to provide 2 important facilities: i) Trace & ii) Breakpoints

i) Trace
• When a processor is operating in trace-mode, an exception occurs after execution of every instruction (using debugging-program as ESR).
• Debugging-program enables user to examine contents of registers, memory-locations and so on.
• On return from debugging-program, next instruction in program being debugged is executed, then debugging-program is activated again.
• The trace exception is disabled during the execution of the debugging-program.

ii) Breakpoints
• Here, the program being debugged is interrupted only at specific points selected by user.

• An instruction called Trap (or Software interrupt) is usually provided for this purpose.
• When program is executed & reaches breakpoint, the user can examine memory & register contents.

3. Privilege Exception

• To protect OS from being corrupted by user-programs, **Privileged Instructions** are executed only while processor is in supervisor-mode.
• For e.g.
When processor runs in user-mode, it will not execute instruction that change priority of processor.
• An attempt to execute privileged-instruction will produce a **Privilege Exception**.
• As a result, processor switches to supervisor-mode & begins to execute an appropriate routine in OS.

DIRECT MEMORY ACCESS (DMA)

• The transfer of a block of data directly b/w an external device & main-memory w/o continuous involvement by processor is called DMA.

• DMA controller

→ is a control circuit that performs DMA transfers (Figure 8.13).
→ is a part of the I/O device interface.
→ performs the functions that would normally be carried out by processor.

• While a DMA transfer is taking place, the processor can be used to execute another program.

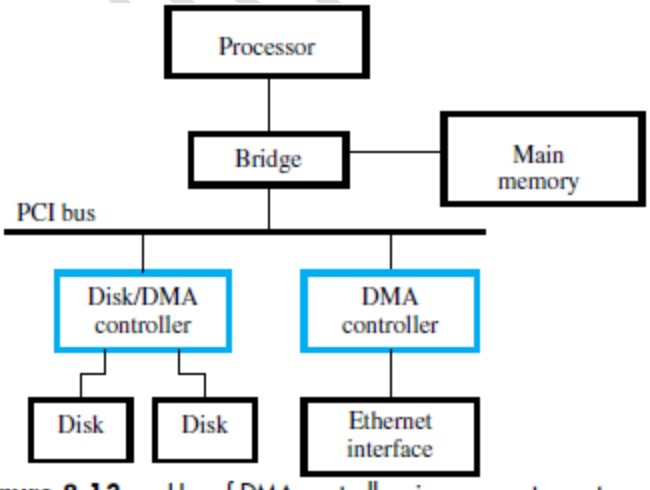


Figure 8.13 Use of DMA controllers in a computer system.

DMA interface has three registers (Figure 8.12):

1) First register is used for storing starting-address.

2) Second register is used for storing word-count.

3) Third register contains status- & control-flags.

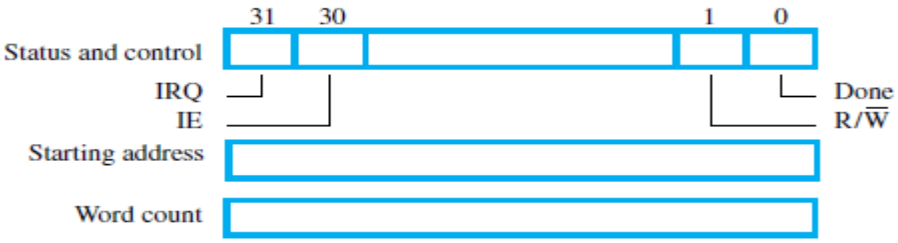


Figure 8.12 Typical registers in a DMA controller.

The R/W bit determines direction of transfer.

If R/W=1, controller performs a read-operation (i.e. it transfers data from memory to I/O), Otherwise, controller performs a write-operation (i.e. it transfers data from I/O to memory).

• If Done=1, the controller
→ has completed transferring a block of data and
→ is ready to receive another command. (IE □□Interrupt Enable).
• If IE=1, controller raises an interrupt after it has completed transferring a block of data.
• If IRQ=1, controller requests an interrupt.
• Requests by DMA devices for using the bus are always given higher priority than processor requests.

• There are 2 ways in which the DMA operation can be carried out:

1) Processor originates most memory-access cycles.
DMA controller is said to "steal" memory cycles from processor. Hence, this technique is usually called **Cycle Stealing**.

2) DMA controller is given exclusive access to main-memory to transfer a block of data without any interruption. This is known as **Block Mode** (or burst mode).

Q Write a short note on any 1 bus arbitration scheme.**June 2014,July 2016**

BUS ARBITRATION
• The device that is allowed to initiate data-transfers on bus at any given time is called **bus-master**.
• There can be only one bus-master at any given time.
• **Bus Arbitration** is the process by which
→ next device to become the bus-master is selected &
→ bus-mastership is transferred to that device.
• The two approaches are:

1) Centralized Arbitration: A single bus-arbiter performs the required arbitration.
2) Distributed Arbitration: All devices participate in selection of next bus-master.

• A conflict may arise if both the processor and a DMA controller or two DMA controllers try to use the bus at the same time to access the main-memory.

• To resolve this, an arbitration procedure is implemented on the bus to coordinate the activities of all devices requesting memory transfers.

• The bus arbiter may be the processor or a separate unit connected to the bus.

CENTRALIZED ARBITRATION

• A single bus-arbiter performs the required arbitration (Figure: 4.20).
• Normally, processor is the bus-master.
• Processor may grant bus-mastership to one of the DMA controllers.
• A DMA controller indicates that it needs to become bus-master by activating BR line.
• The signal on the BR line is the logical OR of bus-requests from all devices connected to it.
• Then, processor activates BG1 signal indicating to DMA controllers to use bus when it becomes free.
• BG1 signal is connected to all DMA controllers using a daisy-chain arrangement.
• If DMA controller-1 is requesting the bus,
Then, DMA controller-1 blocks propagation of grant-signal to other devices.
Otherwise, DMA controller-1 passes the grant downstream by asserting BG2.
• Current bus-master indicates to all devices that it is using bus by activating BBSY line.
• The bus-arbiter is used to coordinate the activities of all devices requesting memory transfers.
• Arbiter ensures that only 1 request is granted at any given time according to a priority scheme.
(BR : Bus-Request, BG : Bus-Grant, BBSY : Bus Busy).

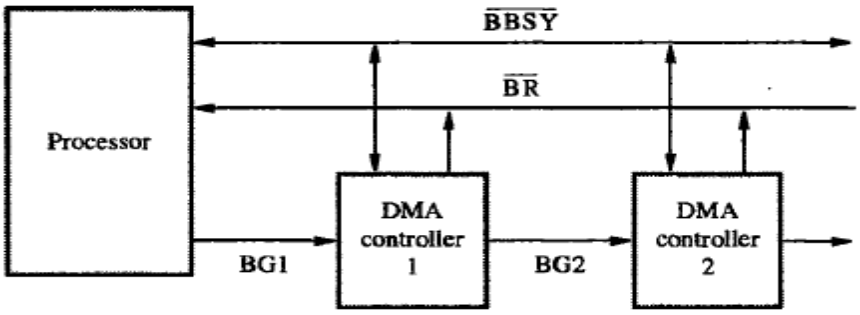


Figure 4.20 A simple arrangement for bus arbitration using a daisy chain.

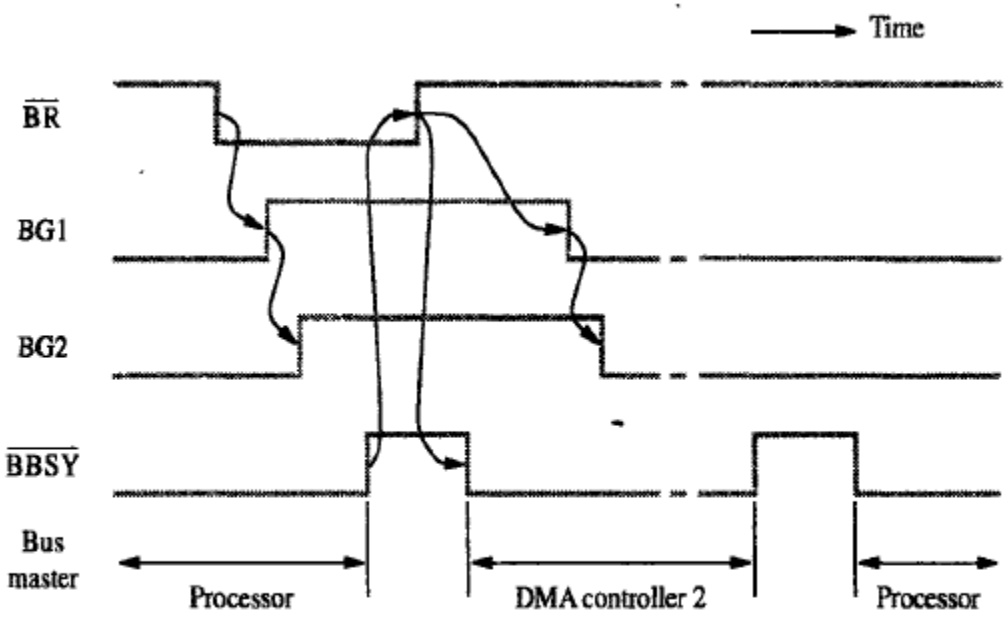


Figure 4.21 Sequence of signals during transfer of bus mastership for the devices in Figure 4.20.

• The timing diagram shows the sequence of events for the devices connected to the processor.
• DMA controller-2
→ requests and acquires bus-mastership and
→ later releases the bus. (Figure: 4.21).
• After DMA controller-2 releases the bus, the processor resources bus-mastership.

DISTRIBUTED ARBITRATION

• All device participate in the selection of next bus-master (Figure 4.22).
• Each device on bus is assigned a 4-bit identification number (ID).
• When 1 or more devices request bus, they
→ assert Start-Arbitration signal &
→ place their 4-bit ID numbers on four open-collector lines *ARB*0 through *ARB*3 .
• A winner is selected as a result of interaction among signals transmitted over these lines.
• Net-outcome is that the code on 4 lines represents request that has the highest ID number.

• **Advantage:**
This approach offers higher reliability since operation of bus is not dependent on any single device.

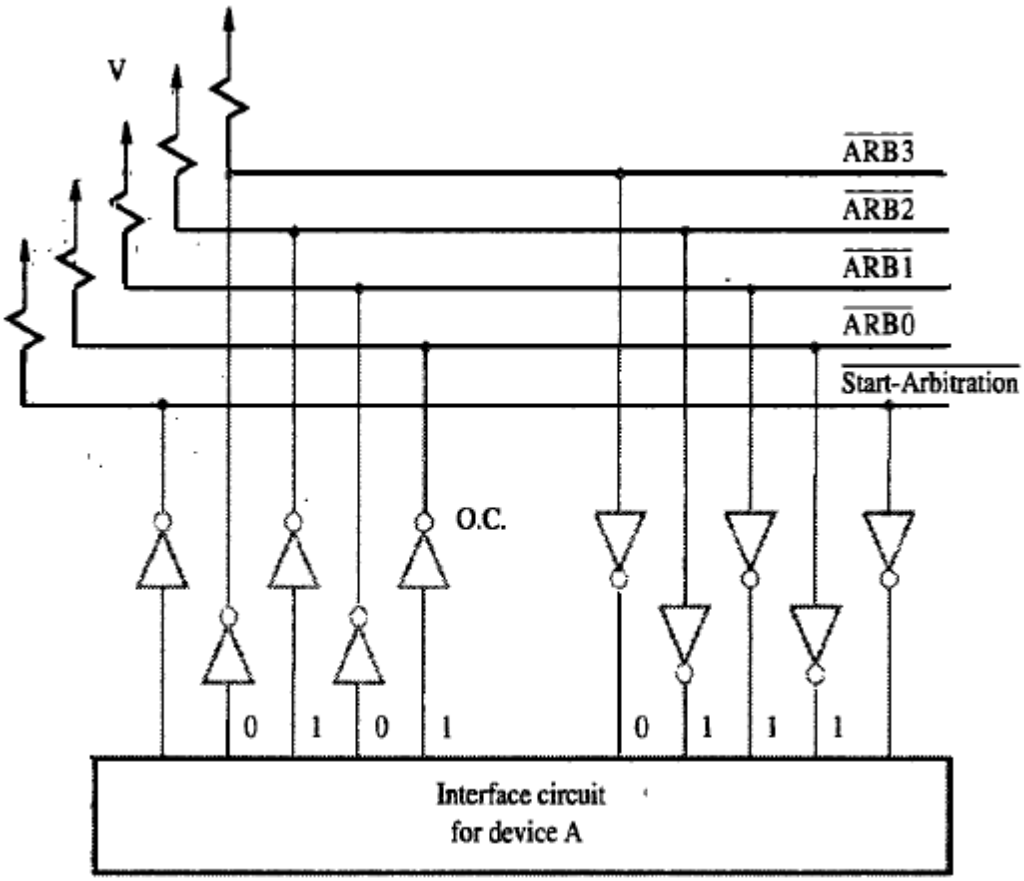


Figure 4.22 A distributed arbitration scheme.

For example:
Assume 2 devices A & B have their ID 5 (0101), 6 (0110) and their code is 0111. Each device compares the pattern on the arbitration line to its own ID starting from MSB. If the device detects a difference at any bit position, it disables the drivers at that bit position. Driver is disabled by placing ”0” at the input of the driver.

In e.g. “A” detects a difference in line ARB1, hence it disables the drivers on lines ARB1 & ARB0. This causes pattern on arbitration-line to change to 0110. This means that “B” has won contention.

BUS
• Bus
→ is used to inter-connect main-memory, processor & I/O-devices
→ includes lines needed to support interrupts & arbitration.
• Primary function: To provide a communication-path for transfer of data.

• **Bus protocol** is set of rules that govern the behavior of various devices connected to the buses.
• Bus-protocol specifies parameters such as:
→ asserting control-signals

→ timing of placing information on bus
→ rate of data-transfer.

• A typical bus consists of 3 sets of lines:
1) Address,
2) Data &
3) Control lines.

• Control-signals
→ specify whether a read or a write-operation is to be performed.
→ carry timing information i.e. they specify time at which I/O-devices place data on the bus.

• R/W line specifies
→ read-operation when R/W=1.
→ write-operation when R/W=0.

• During data-transfer operation,
One device plays the role of a bus-master.
Master-device initiates the data-transfer by issuing read/write command on the bus.
The device addressed by the master is called as Slave.

• Two types of Buses: 1) Synchronous and 2) Asynchronous.

SYNCHRONOUS BUS
• All devices derive timing-information from a common clock-line.
• Equally spaced pulses on this line define equal time intervals.
• During a ”bus cycle“, one data-transfer can take place.

A sequence of events during a read-operation
• At time t0, the master (processor)
→ places the device-address on address-lines &
→ sends an appropriate command on control-lines (Figure 7.3).
• The command will
→ indicate an input operation &
→ specify the length of the operand to be read.
• Information travels over bus at a speed determined by physical & electrical characteristics.
• Clock pulse width(t1-t0) must be longer than max. propagation-delay b/w devices connected to bus.
• The clock pulse width should be long to allow the devices to decode the address & control signals.
• The slaves take no action or place any data on the bus before t1.
• Information on bus is unreliable during the period t0 to t1 because signals are changing state.
• Slave places requested input-data on data-lines at time t1.
• At end of clock cycle (at time t2), master strobes (captures) data on data-lines into its input-buffer
• For data to be loaded correctly into a storage device,
data must be available at input of that device for a period greater than setup-time of device.

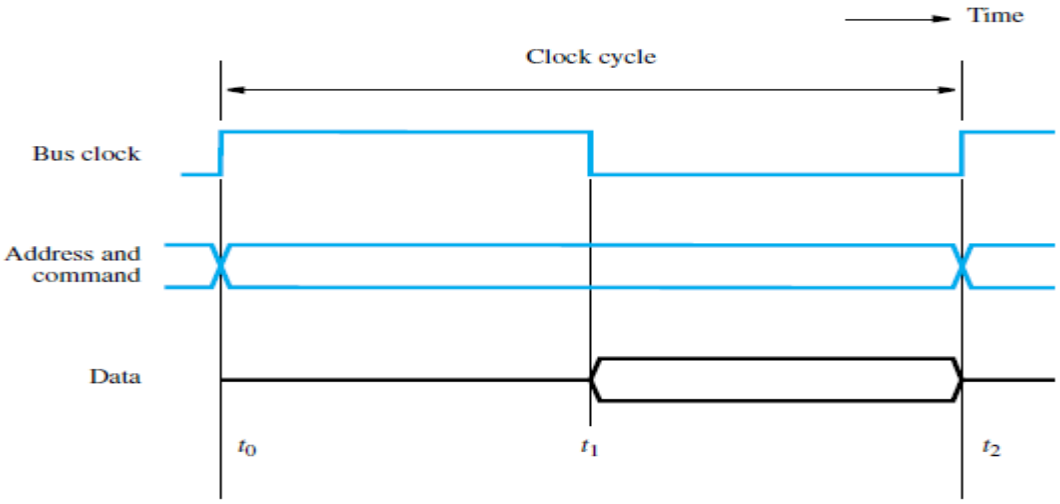


Figure 7.3 Timing of an input transfer on a synchronous bus.

A Detailed Timing Diagram for the Read-operation

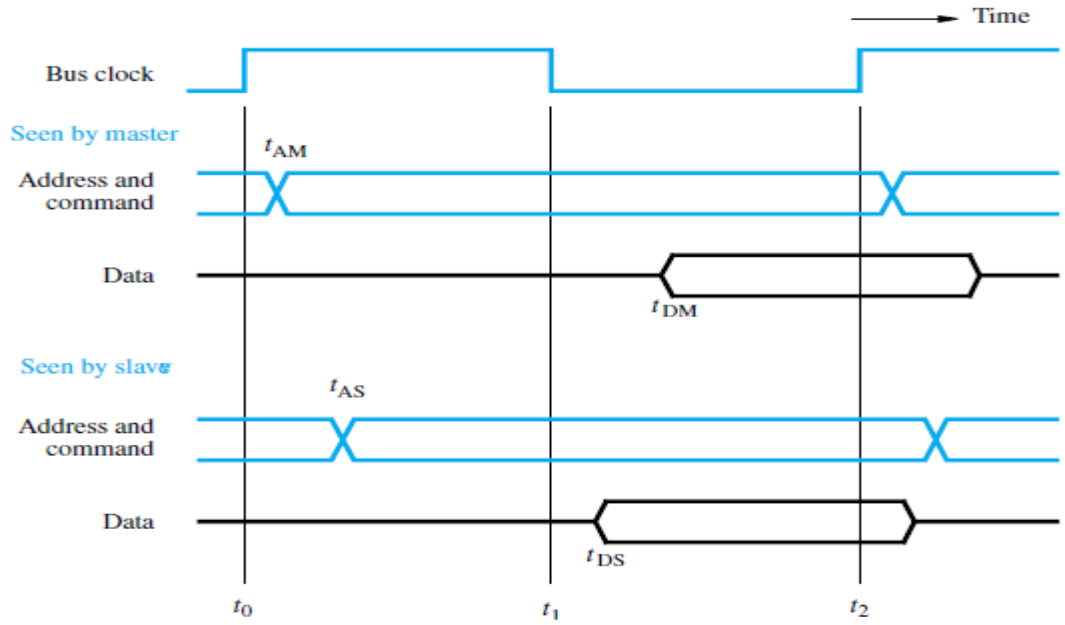


Figure 7.4 A detailed timing diagram for the input transfer of Figure 7.3.

• The picture shows two views of the signal except the clock (Figure 7.4).
• One view shows the signal seen by the master & the other is seen by the salve.
• Master sends the address & command signals on the rising edge at the beginning of clock period (t0).
• These signals do not actually appear on the bus until tam.
• Sometimes later, at tAS the signals reach the slave.
• The slave decodes the address.
• At t1, the slave sends the requested-data.
• At t2, the master loads the data into its input-buffer.
• Hence the period t2, tDM is the setup time for the master"s input-buffer.
• The data must be continued to be valid after t2, for a period equal to the hold time of that buffers.

Disadvantages

• The device does not respond.
• The error will not be detected.

Multiple Cycle Transfer for Read-operation

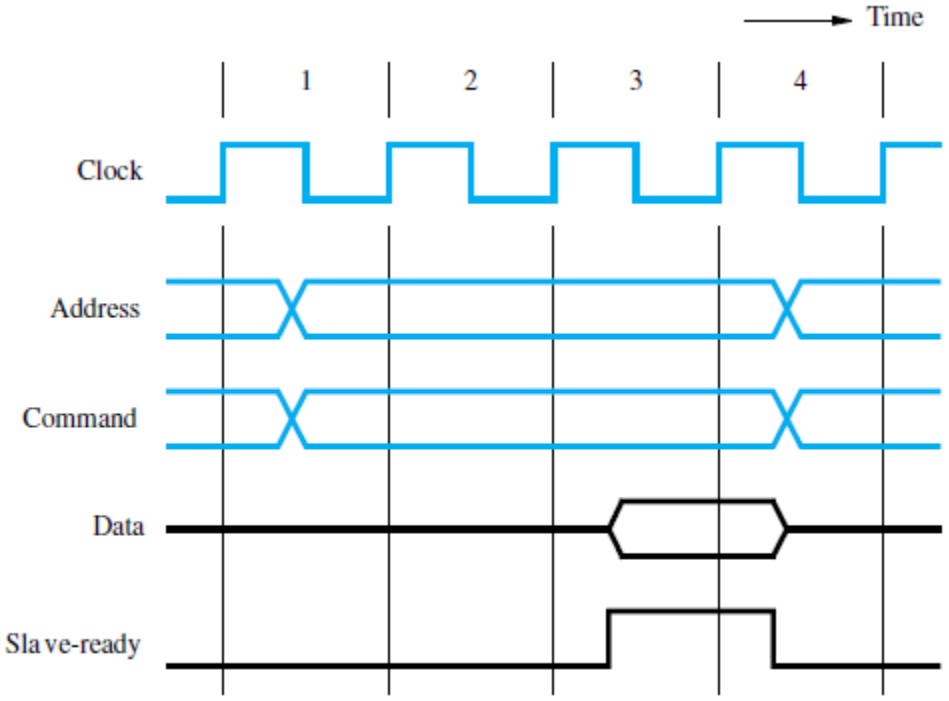


Figure 7.5 An input transfer using multiple clock cycles.

• During, clock cycle-1, master sends address/command info the bus requesting a "read" operation.
• The slave receives & decodes address/command information (Figure 7.5).
• At the active edge of the clock i.e. the beginning of clock cycle-2, it makes accession to respond immediately.
• The data become ready & are placed in the bus at clock cycle-3.
• At the same times, the slave asserts a control signal called **slave-ready**.
• The master strobes the data to its input-buffer at the end of clock cycle-3.
• The bus transfer operation is now complete.
• And the master sends a new address to start a new transfer in clock cycle4.
• The slave-ready signal is an acknowledgement from the slave to the master.

ASYNCHRONOUS BUS

• This method uses handshake-signals between master and slave for coordinating data-transfers.
• There are 2 control-lines:

1) Master-Ready (MR) is used to indicate that master is ready for a transaction.
2) Slave-Ready (SR) is used to indicate that slave is ready for a transaction.

The Read Operation proceeds as follows:

• At t0, master places address/command information on bus.

• At t1, master sets MR-signal to 1 to inform all devices that the address/command-info is ready.

MR-signal =1 causes all devices on the bus to decode the address. The delay t1 – t0 is intended to allow for any skew that may occurs on the bus. Skew occurs when 2 signals transmitted from 1 source arrive at destination at different time Therefore, the delay t1 – t0 should be larger than the maximum possible bus skew.

• At t2, slave
→ performs required input-operation &
→ sets SR signal to 1 to inform all devices that it is ready (Figure 7.6).

• At t3, SR signal arrives at master indicating that the input-data are available on bus.
• At t4, master removes address/command information from bus.
• At t5, when the device-interface receives the 1-to-0 transition of MR signal, it removes data and SR signal from the bus. This completes the input transfer.

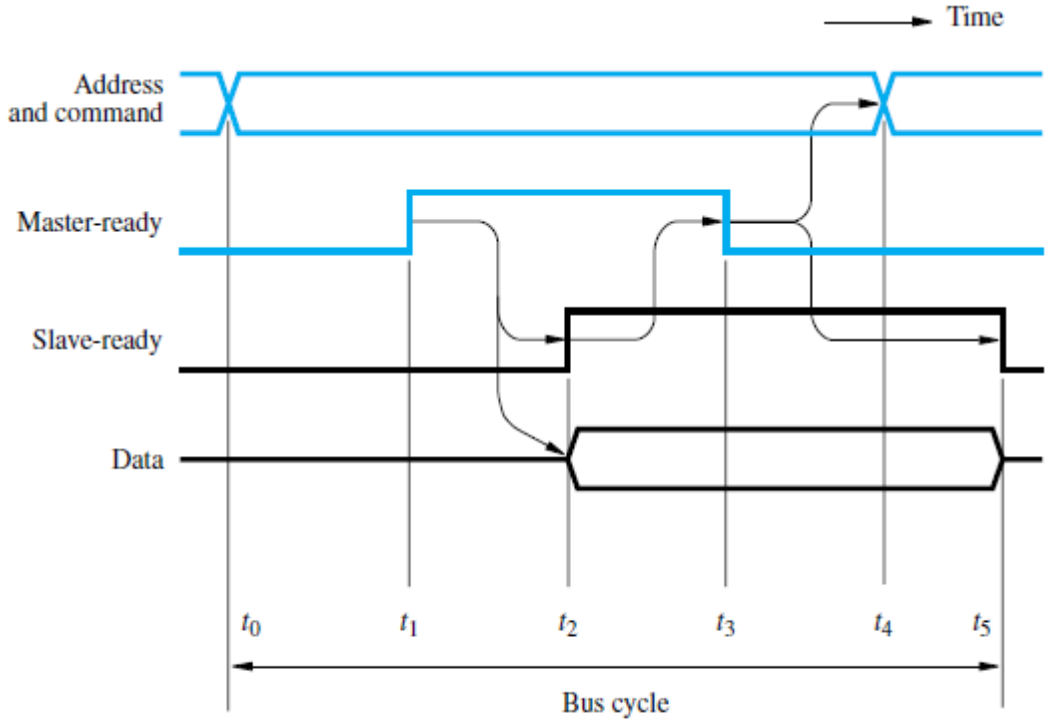


Figure 7.6 Handshake control of data transfer during an input operation.

• A change of state is one signal is followed by a change is the other signal. Hence this scheme is called as **Full Handshake**.

• **Advantage:** It provides the higher degree of flexibility and reliability.

INTERFACE-CIRCUITS

• An **I/O Interface** consists of the circuitry required to connect an I/O device to a computer-bus.
• On one side of the interface, we have bus signals.
On the other side, we have a data path with its associated controls to transfer data between the interface and the I/O device known as **port**.

• Two types are:
1. Parallel Port transfers data in the form of a number of bits (8 or 16) simultaneously to or from the device.

2. Serial Port transmits and receives data one bit at a time.
• Communication with the bus is the same for both formats.
• The conversion from the parallel to the serial format, and vice versa, takes place inside the interfacecircuit.
• In parallel-port, the connection between the device and the computer uses
→ a multiple-pin connector and
→ a cable with as many wires.
• This arrangement is suitable for devices that are physically close to the computer.
• In serial port, it is much more convenient and cost-effective where longer cables are needed.

Functions of I/O Interface
1) Provides a storage buffer for at least one word of data.
2) Contains status-flags that can be accessed by the processor to determine whether the buffer is full or empty.
3) Contains address-decoding circuitry to determine when it is being addressed by the processor.
4) Generates the appropriate timing signals required by the bus control scheme.
5) Performs any format conversion that may be necessary to transfer data between the bus and the I/O device (such as parallel-serial conversion in the case of a serial port).

Q Draw the hardware components needed for connecting a keyboard to a process and explain in brief. **Jan 2014**

PARALLEL-PORT
KEYBOARD INTERFACED TO PROCESSOR

Input Interface
Figure 7.10 shows a circuit that can be used to connect a keyboard to a processor. There are only two registers: a data register, DATAIN , and a status register, SIN

A typical keyboard consists of mechanical switches that are normally open. When a key is pressed, its switch closes and establishes a path for an electrical signal. This signal is detected by an encoder circuit that generates theASCII code for the corresponding character. A difficulty with such mechanical pushbutton switches is that the contacts *bounce* when a key is pressed, resulting in the electrical connection being made then broken several times before the switch settles in the closed position. Although bouncing may last only one or two milliseconds, this is long enough for the computer to erroneously interpret a single pressing of a key as the key being pressed and released several times. The effect of bouncing can be eliminated using a simple debouncing circuit, which could be part of the keyboard hardware or may be incorporated in the encoder circuit. Alternatively, switch bouncing can be dealt with in software.

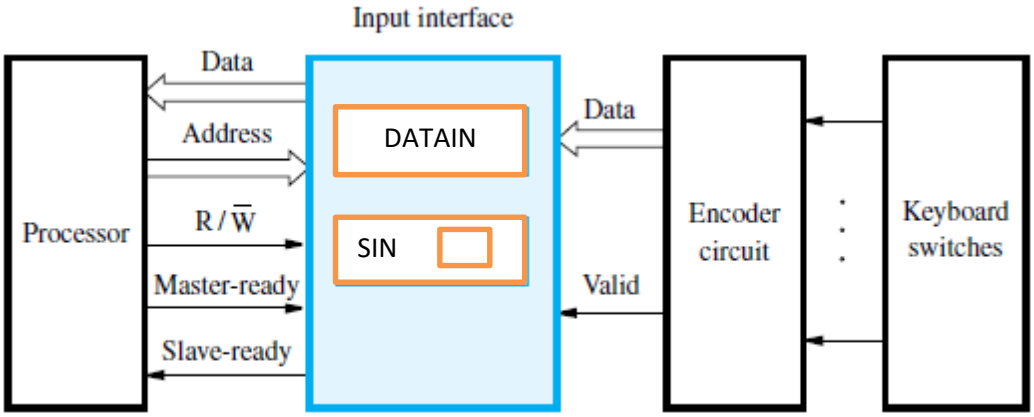


Figure 7.10 Keyboard to processor connection.

• The output of the encoder consists of
→ bits representing the encoded character and
→ one signal called **valid**, which indicates the key is pressed.

• The information is sent to the interface-circuits (Figure 7.10).
• Interface-circuits contain
1) Data register DATAIN &
2) Status-flag SIN.

• When a key is pressed, the Valid signal changes from 0 to1.
Then, SIN=1 when ASCII code is loaded into DATAIN.
SIN = 0 when processor reads the contents of the DATAIN.

• The interface-circuit is connected to the asynchronous bus.
• Data transfers on the bus are controlled using the handshake signals:
1) Master ready &
2) Slave ready.

INPUT-INTERFACE-CIRCUIT

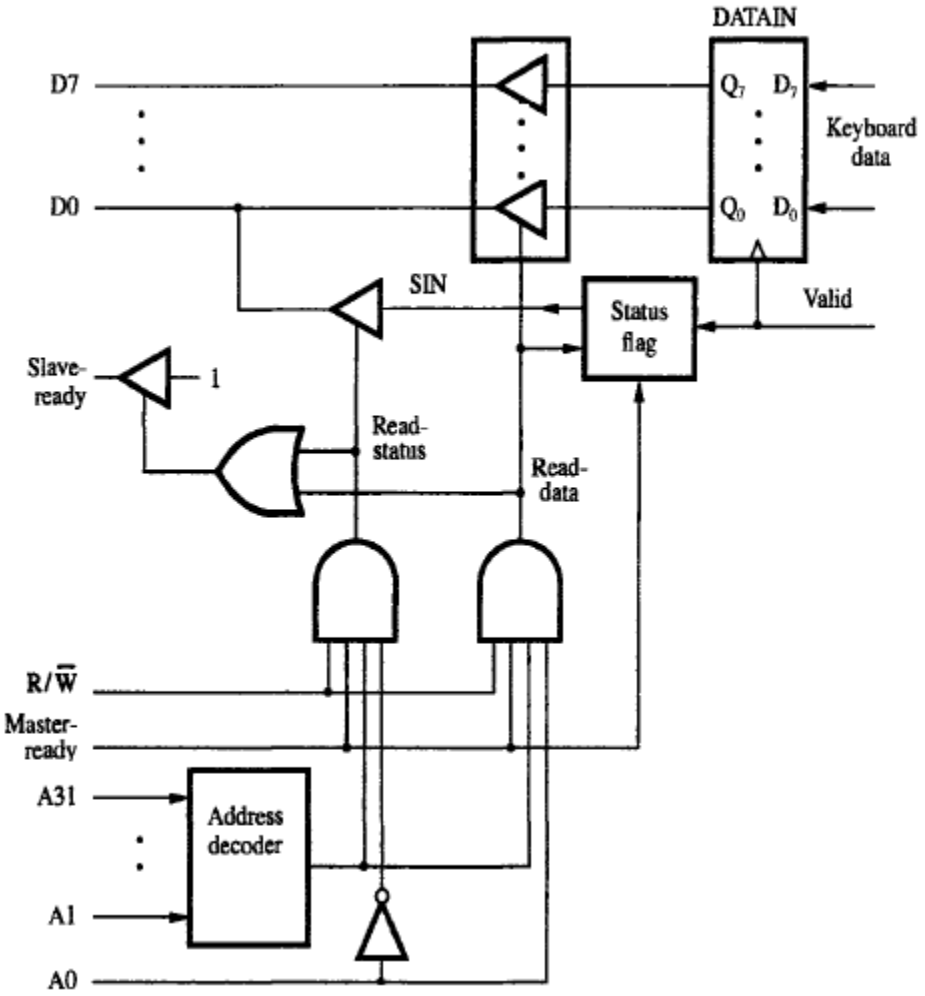


Figure 4.29: Input-interface-circuit

• Output-lines of DATAIN are connected to the data-lines of bus by means of 3-state drivers (Fig 4.29).

• Drivers are turned on when
→ processor issues a read signal and
→ address selects DATAIN.

• SIN signal is generated using a status-flag circuit (Figure 4.30).
SIN signal is connected to line D0 of the processor-bus using a 3-state driver.

• Address-decoder selects the input-interface based on bits A1 through A31.

• Bit A0 determines whether the status or data register is to be read, when Master-ready is active.

• Processor activates the Slave-ready signal, when either the Read-status or Read-data is equal to 1.

• Keyboard is connected to a processor using a parallel-port.

• Processor uses
→ memory-mapped I/O and
→ asynchronous bus protocol.

• On the processor-side of the interface, we have:
→ Data-lines
→ Address-lines
→ Control or R/W line
→ Master-Ready signal and
→ Slave-Ready signal.

• On the keyboard-side of the interface, we have:
→ Encoder-circuit which generates a code for the key pressed.
→ Debouncing-circuit which eliminates the effect of a key.
→ Data-lines which contain the code for the key.
→ Valid line changes from 0 to 1 when the key is pressed. This causes the code to be loaded into DATAIN and SIN to be set to 1.

PRINTER INTERFACED TO PROCESSOR

Let us now consider the output interface shown in Figure 7.13, which can be used to connect an output device such as a display. When the display is ready to accept a character, it asserts its Ready signal, which causes the SOUT register to be set to 1. When the I/O routine checks SOUT and finds it equal to 1, it sends a character to DATAOUT. This clears the SOUT flag to 0 and sets the

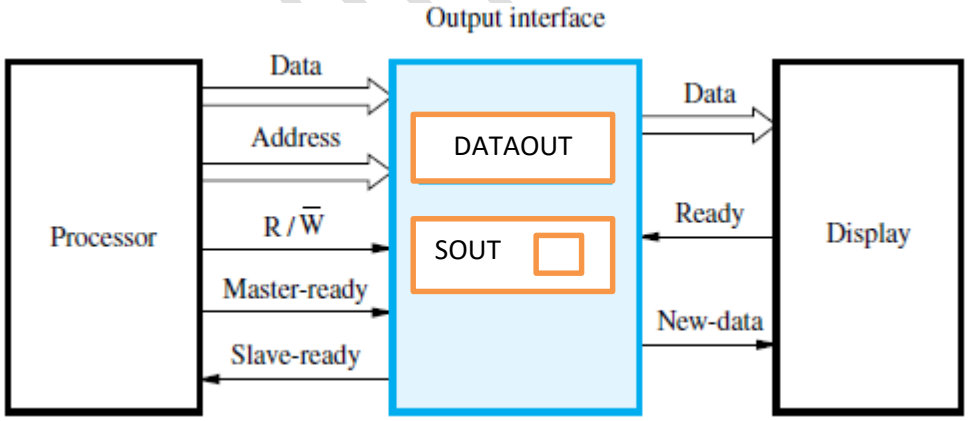


Figure 7.13 Display to processor connection.

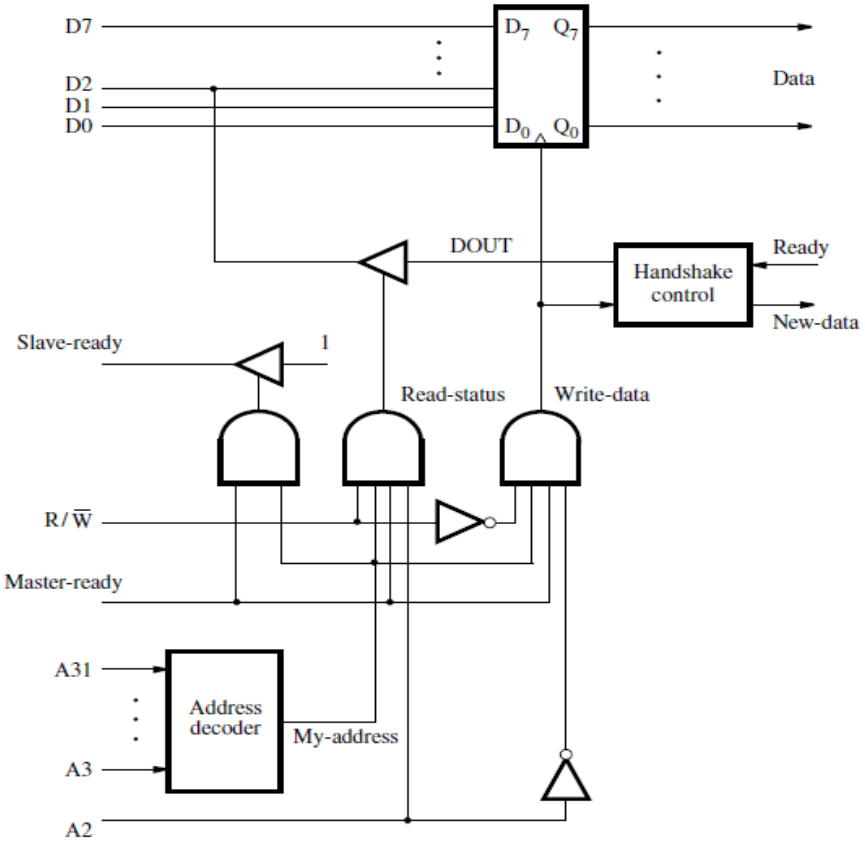


Figure 7.14 An output interface circuit.

Q Explain with a block diagram a general 8 bit serial interface.

7.4.2 Serial Interface

A serial interface is used to connect the processor to I/O devices that transmit data one bit at a time. Data are transferred in a bit-serial fashion on the device side and in a bit-parallel fashion on the processor side. The transformation between the parallel and serial formats is achieved with shift registers that have parallel access capability. A block diagram of a typical serial interface is shown in Figure 7.15. The input shift register accepts bit-serial input from the I/O device. When all 8 bits of data have been received, the contents of this shift register are loaded in parallel into the DATAIN register. Similarly, output data in the DATAOUT register are transferred to the output shift register, from which the bits are shifted out and sent to the I/O device.

The part of the interface that deals with the bus is the same as in the parallel interface described earlier. Two status flags, which we will refer to as SIN and SOUT, are maintained by the Status and control block. The SIN flag is set to 1 when new data are loaded into DATAIN from the shift register, and cleared to 0 when these data are read by the processor. The SOUT flag indicates whether the DATAOUT register is available. It is cleared to 0 when the processor writes new data into DATAOUT and set to 1 when data are transferred from DATAOUT to the output shift register.

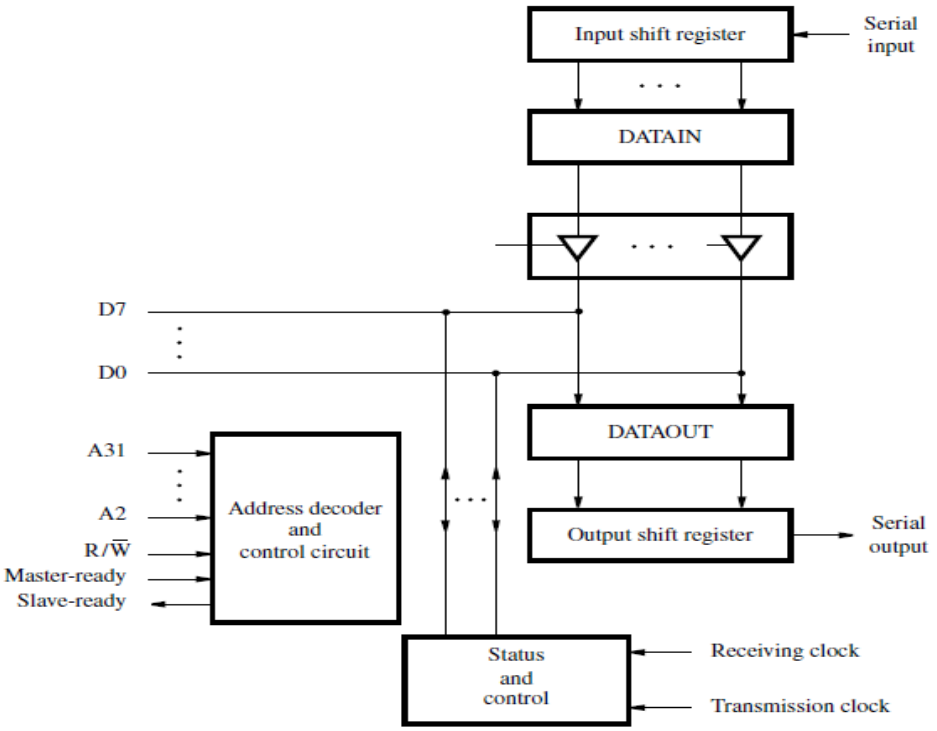


Figure 7.15 A serial interface.

Q Explain with a block diagram a general 8 bit parallel interface. **July 2015**

GENERAL 8 BIT PARALLEL PROCESSING

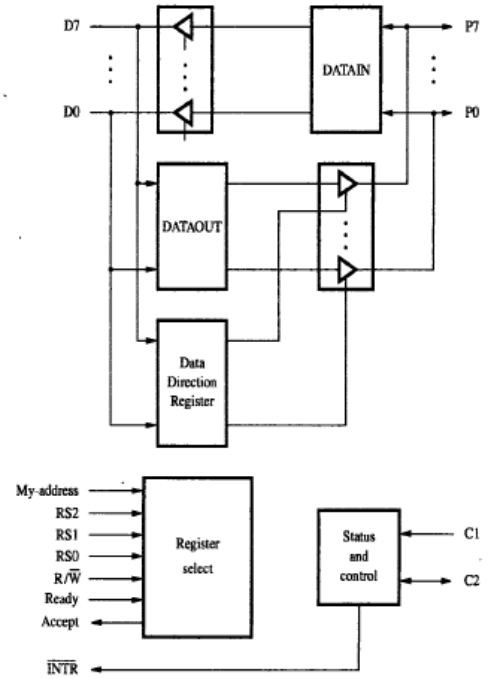


Figure 4.34: General 8 bit parallel interface

• Data-lines **P7** through **PO** can be used for either input or output purposes (Figure 4.34).
• For increased flexibility,
→ some lines can be used as inputs and
→ some lines can be used as outputs.
• The **DATAOUT** register is connected to data-lines via 3-state drivers that are controlled by a **DDR**.
• The processor can write any 8-bit pattern into DDR. (DDR :Data Direction Register).
• If DDR=1,
Then, data-line acts as an output-line;
Otherwise, data-line acts as an input-line.
• Two lines, **C1** and **C2** are used to control the interaction between interface-circuit and I/0 device.
Two lines, C1 and C2 are also programmable.
• Line C2 is bidirectional to provide different modes of signaling, including the handshake.
• The **Ready** and **Accept** lines are the handshake control lines on the processor-bus side. Hence, the Ready and Accept lines can be connected to Master-ready and Slave-ready.
• The input signal **My-address** should be connected to the output of an address-decoder. The address-decoder recognizes the address assigned to the interface.
• There are 3 register select lines: **RS0-RS2**.
Three register select lines allows up to eight registers in the interface.
• An interrupt-request **INTR** is also provided.
INTR should be connected to the interrupt-request line on the computer-bus.

STANDARD I/O INTERFACE
• Consider a computer system using different interface standards.
• Let us look in to Processor bus and Peripheral Component Interconnect (PCI) bus (Figure 4.38).
• These two buses are interconnected by a circuit called **Bridge**.
• The bridge translates the signals and protocols of one bus into another.
• The bridge-circuit introduces a small delay in data transfer between processor and the devices.

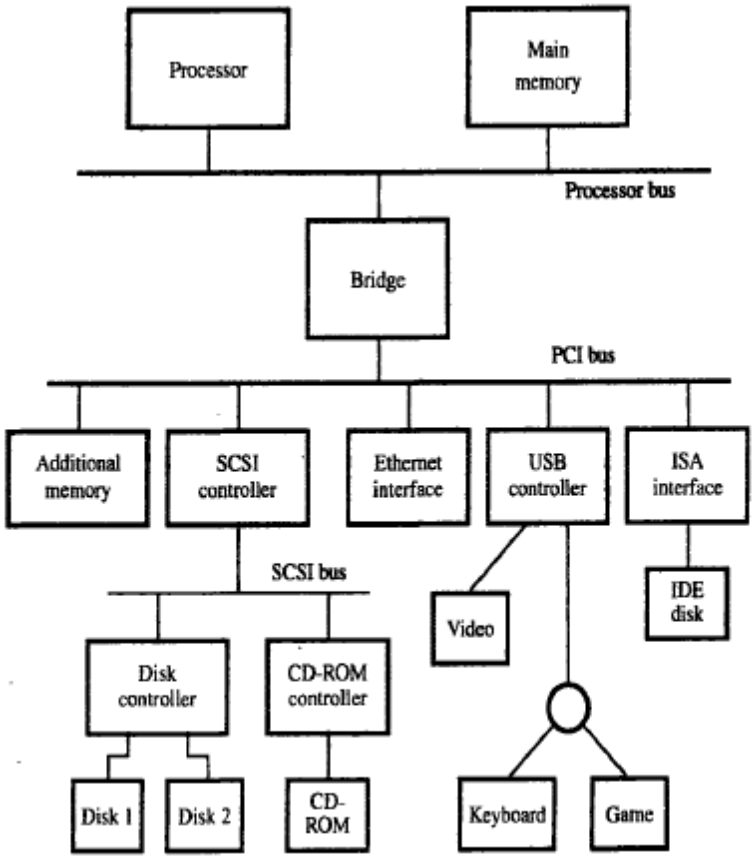


Figure 4.38 An example of a computer system using different interface standards.

• The 3 major standard I/O interfaces are:
1) PCI (Peripheral Component Interconnect)
2) SCSI (Small Computer System Interface)
3) USB (Universal Serial Bus)
• PCI defines an expansion bus on the motherboard.
• SCSI and USB are used for connecting additional devices both inside and outside the computer-box.
• SCSI bus is a high speed parallel bus intended for devices such as disk and video display.
• USB uses a serial transmission to suit the needs of equipment ranging from keyboard to game control to internal connection.
• IDE (Integrated Device Electronics) disk is compatible with ISA which shows the connection to an Ethernet.

PCI
• PCI is developed as a low cost bus that is truly processor independent.
• PCI supports high speed disk, graphics and video devices.
• PCI has plug and play capability for connecting I/O devices.
• To connect new devices, the user simply connects the device interface board to the bus.

DATA TRANSFER IN PCI
• The data are transferred between cache and main-memory.
• The data is a sequence of words which are stored in successive memory-locations.
• During **read-operation**,

When the processor specifies an address, the memory responds by sending a sequence of data-words from successive memory-locations.

• During **write-operation**,
When the processor sends an address, a sequence of data-words is written into successive memory-locations.

• PCI supports read and write-operation.
• A read/write-operation involving a single word is treated as a burst of length one.

• PCI has 3 address-spaces. They are
1) Memory address-space
2) I/O address-space &
3) Configuration address-space.

• I/O Address-space Intended for use with processor.
Configuration space Intended to give PCI, its plug and play capability.

• **PCI Bridge** provides a separate physical connection to main-memory.
• The master maintains the address information on the bus until data-transfer is completed.
• At any time, only one device acts as **Bus-Master**.
• A master is called "initiator" which is either processor or DMA.
• The addressed-device that responds to read and write commands is called a **Target.**
• A complete transfer operation on the bus, involving an address and burst of data is called a **transaction.**

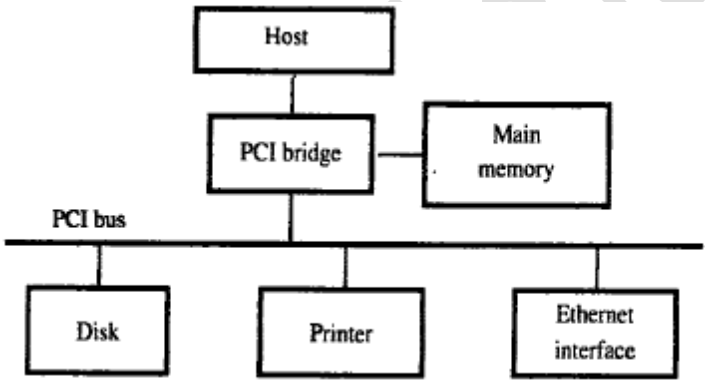


Figure 4.39 Use of a PCI bus in a computer system.

Q With the help of a data transfer signals explain how a real operation is performed usingPCI bus. **Dec 2015**
Q11.In a computer system why a PCI bus is used? With a neat sketch, explain how the read operation is performed along with the role of IRDY#/TRDY# on the PCI bus **Dec 2014 ,July 2016.**

Table 7.1 Data transfer signals on the PCI bus.

Name	Function
CLK	A 33-MHz or 66-MHz clock
FRAME#	Sent by the initiator to indicate the duration of a transmission
AD	32 address/data lines, which may be optionally increased to 64
C/BE#	4 command/byte-enable lines (8 for a 64-bit bus)
IRDY#, TRDY#	Initiator-ready and Target-ready signals
DEVSEL#	A response from the device indicating that it has recognized its address and is ready for a data transfer transaction
IDSEL#	Initialization Device Select

• Individual word transfers are called "**phases**'.

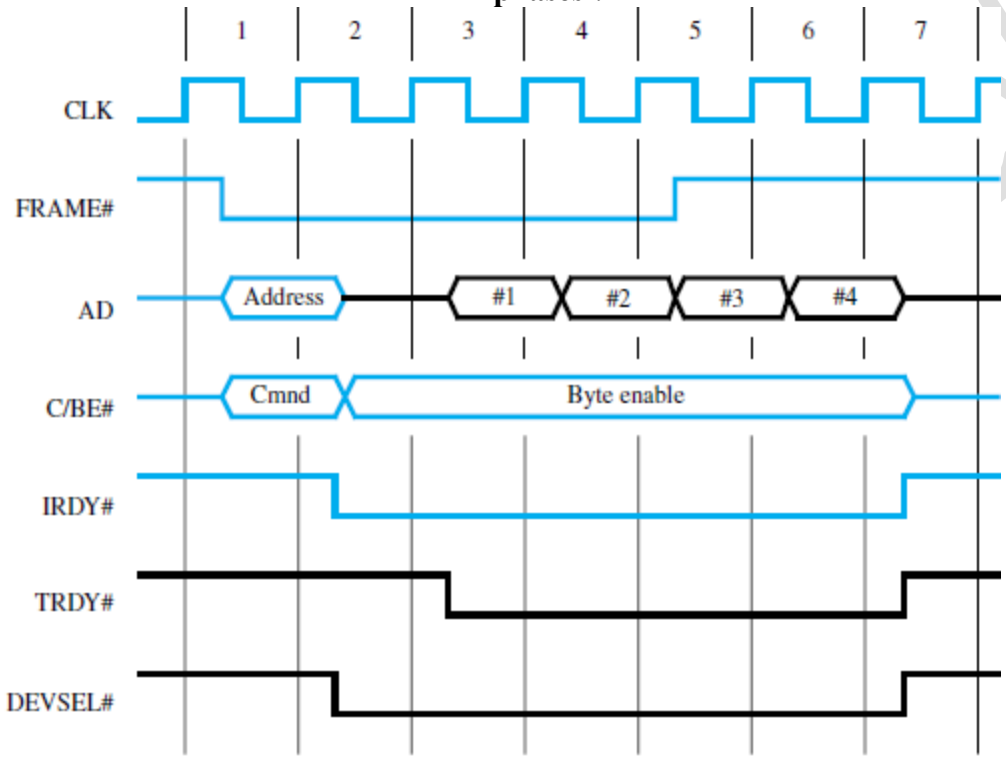


Figure 7.19 A Read operation on the PCI bus.

• During Clock cycle-1,
The processor a
→ asserts FRAME# to indicate the beginning of a transaction;
→ sends the address on AD lines and command on C/BE# Lines.

• During Clock cycle-2,
The processor removes the address and disconnects its drives from AD lines.
Selected target
→ enables its drivers on AD lines and
→ fetches the requested-data to be placed on bus.
Selected target
→ asserts DEVSEL# and
→ maintains it in asserted state until the end of the transaction.
C/BE# is
→ used to send a bus command and it is

→ used for different purpose during the rest of the transaction.

• During Clock cycle-3,
The initiator asserts IRDY# to indicate that it is ready to receive data.
If the target has data ready to send then it asserts TRDY#. In our eg, the target sends 3 more words of data in clock cycle 4 to 6.

• During Clock cycle-5
The indicator uses FRAME# to indicate the duration of the burst, since it read 4 words, the initiator negates FRAME# during clock cycle 5.

• During Clock cycle-7,
After sending 4th word, the target
→ disconnects its drivers and
→ negates DEVSEL# during clock cycle 7.

DEVICE CONFIGURATION OF PCI
• The PCI has a configuration ROM that stores information about that device.
• The configuration ROM's of all devices are accessible in the configuration address-space.
• The initialization software read these ROM's whenever the system is powered up or reset.
• In each case, it determines whether the device is a printer, keyboard or disk controller.
• Devices are assigned address during initialization process.
• Each device has an input signal called IDSEL# (Initialization device select) which has 21 address lines (AD11 to AD31).
• During configuration operation,
The address is applied to AD input of the device and
The corresponding AD line is set to 1 and all other lines are set to 0.

AD11 - AD31
Upper address-line

A0 - A10

Lower address-line: Specify the type of the operation and to access the content of device configuration ROM.

• The configuration software scans all 21 locations. PCI bus has interrupt-request lines.
• Each device may requests an address in the I/O space or memory space

SCSI Bus

• SCSI stands for Small Computer System Interface.
• SCSI refers to the standard bus which is defined by ANSI (American National Standard Institute).
• SCSI bus the several options. It may be,

Narrow bus	It has 8 data-lines & transfers 1 byte at a time.
Wide bus	It has 16 data-lines & transfer 2 byte at a time.
Single-Ended Transmission	Each signal uses separate wire.
HVD (High Voltage Differential)	It was 5v (TTL cells)
LVD (Low Voltage Differential)	It uses 3.3v

• Because of these various options, SCSI connector may have 50, 68 or 80 pins. The data transfer rate ranges from 5MB/s to 160MB/s 320Mb/s, 640MB/s. The transfer rate depends on,

1) Length of the cable
2) Number of devices connected.
• To achieve high transfer rate, the bus length should be 1.6m for SE signaling and 12m for LVD signaling.
• The SCSI bus us connected to the processor-bus through the SCSI controller. The data are stored on a disk in blocks called sectors.
Each sector contains several hundreds of bytes. These data will not be stored in contiguous memory-location.
• SCSI protocol is designed to retrieve the data in the first sector or any other selected sectors.
• Using SCSI protocol, the burst of data are transferred at high speed.
• The controller connected to SCSI bus is of 2 types. They are1) Initiator * 2) Target

1) Initiator
It has the ability to select a particular target & to send commands specifying the operation to be performed. They are the controllers on the processor side.

2) Target
The disk controller operates as a target.
It carries out the commands it receive from the initiator. The initiator establishes a logical connection with the intended target.

Steps for Read-operation
1) The SCSI controller contends for control of the bus (initiator).
2) When the initiator wins the arbitration-process, the initiator
→ selects the target controller and
→ hands over control of the bus to it.
3) The target starts an output operation. The initiator sends a command specifying the required read operation.
4) The target
→ sends a message to initiator indicating that it will temporarily suspend connection b/w them.
→ then releases the bus.
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.
6. The target
→ transfers the contents of the data buffer to the initiator and
→ then suspends the connection again.
7) The target controller sends a command to the disk drive to perform another seek operation.
8) As the initiator controller receives the data, it stores them into the main-memory using the DMA approach.
9) The SCSI controller sends an interrupt to the processor indicating that the data are now available.

Q List the SCSI bus signals with their functionalities. **Jan 2014**

BUS SIGNALS OF SCSI

• The bus has no address-lines. Instead, it has data-lines to identify the bus-controllers involved in the selection/reselection/arbitration-process.
• For narrow bus, there are 8 possible controllers numbered from 0 to 7. For a wide bus, there are 16 controllers.
• Once a connection is established b/w two controllers, there is no further need for addressing & the data-lines are used to carry the data.

Table 4.4 The SCSI bus signals

Category	Name	Function
Data	–DB(0) to –DB(7)	Data lines: Carry one byte of information during the information transfer phase and identify device during arbitration, selection and reselection phases
	–DB(P)	Parity bit for the data bus
Phase	–BSY	Busy: Asserted when the bus is not free
	–SEL	Selection: Asserted during selection and reselection
Information type	–C/D	Control/Data: Asserted during transfer of control information (command, status or message)
	–MSG	Message: indicates that the information being transferred is a message
Handshake	–REQ	Request: Asserted by a target to request a data transfer cycle
	–ACK	Acknowledge: Asserted by the initiator when it has completed a data transfer operation
Direction of transfer	–I/O	Input/Output: Asserted to indicate an input operation (relative to the initiator)
Other	–ATN	Attention: Asserted by an initiator when it wishes to send a message to a target
	–RST	Reset: Causes all device controls to disconnect from the bus and assume their start-up state

• All signal names are proceeded by minus sign.
• This indicates that the signals are active or that the data-line is equal to 1, when they are in the low voltage state.

PHASES IN SCSI BUS
• The phases in SCSI bus operation are:
1) Arbitration
2) Selection
3) Information transfer
4) Reselection

Q Explain briefly bus arbitration phase in SCSI bus. **June 2015,July 2016**

1) Arbitration
• When the –BSY signal is in inactive state,
→ the bus will be free &
→ any controller can request the use of bus.

• SCSI uses distributed arbitration scheme because each controller may generate requests at the same time.

• Each controller on the bus is assigned a fixed priority.

• When –BSY becomes active, all controllers that are requesting the bus
→ examines the data-lines &
→ determine whether highest priority device is requesting bus at the same time.

• The controller using the highest numbered line realizes that it has won the arbitration-process.

• At that time, all other controllers disconnect from the bus & wait for –BSY to become inactive again.

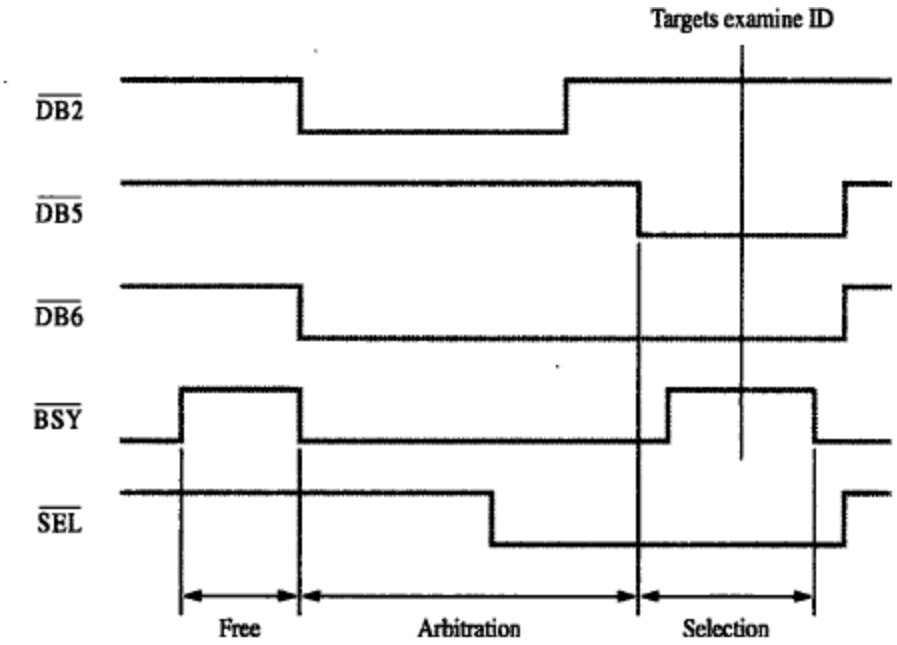


Figure 4.42 Arbitration and selection on the SCSI bus. Device 6 wins arbitration and selects device 2.

2) Information Transfer
• The information transferred between two controllers may consist of
→ commands from the initiator to the target
→ status responses from the target to the initiator or
→ data-transferred to/from the I/0 device.
• Handshake signaling is used to control information transfers, with the target controller taking the role of the bus-master.

3) Selection
• Here, Device
→ wins arbitration and
→ asserts –BSY and –DB6 signals.
• The Select Target Controller responds by asserting –BSY.
• This informs that the connection that it requested is established.

4) Reselection
• The connection between the two controllers has been reestablished, with the target in control of the bus as required for data transfer to proceed.

USB

• USB stands for Universal Serial Bus.
• USB supports 3 speed of operation. They are,
1) Low speed (1.5 Mbps)
2) Full speed (12 mbps) &
3) High speed (480 mbps).
• The USB has been designed to meet the key objectives. They are,
1) Provide a simple, low-cost and easy to use interconnection system.
This overcomes difficulties due to the limited number of I/O ports available on a computer.
2) Accommodate a wide range of data transfer characteristics for I/O devices.
For e.g. telephone and Internet connections
3) Enhance user convenience through a "plug-and-play" mode of operation.
• **Advantage:** USB helps to add many devices to a computer system at any time without opening the computer-box.

Port Limitation

Normally, the system has a few limited ports.
To add new ports, the user must open the computer-box to gain access to the internal expansion bus & install a new interface card.
The user may also need to know to configure the device & the s/w.

Plug & Play

The main objective: USB provides a plug & play capability.
The plug & play feature enhances the connection of new device at any time, while the system is operation.
The system should
→ Detect the existence of the new device automatically.
→ Identify the appropriate device driver s/w.
→ Establish the appropriate addresses.
→ Establish the logical connection for communication.

DEVICE CHARACTERISTICS OF USB

• The kinds of devices that may be connected to a computer cover a wide range of functionality.
• The speed, volume & timing constrains associated with data transfer to & from devices varies significantly.

Eg: 1 Keyboard

Since the event of pressing a key is not synchronized to any other event in a computer system, the data generated by keyboard are called asynchronous.
The data generated from keyboard depends upon the speed of the human operator which is about 100 bytes/sec.

Eg: 2 Microphone attached in a computer system internally/externally

The sound picked up by the microphone produces an analog electric signal, which must be converted into digital form before it can be handled by the computer.
This is accomplished by sampling the analog signal periodically.
The sampling process yields a continuous stream of digitized samples that arrive at regular intervals, synchronized with the sampling clock. Such a stream is called isochronous (i.e.) successive events are separated by equal period of time.

If the sampling rate in „S" samples/sec then the maximum frequency captured by sampling process is s/2.
A standard rate for digital sound is 44.1 KHz.

Q Draw the block diagram of universal bus(USB)structure connected to the host computer Briefly explain all fields of packets that are used for communication between a host and a device connected to an USB port. **June 2014/July 2015**

USB ARCHITECTURE

• To accommodate a large number of devices that can be added or removed at any time, the USB has the tree structure as shown in the figure 7.17.
• Each node of the tree has a device called a **Hub**.
• A hub acts as an intermediate control point between the host and the I/O devices.
• At the root of the tree, a **Root Hub** connects the entire tree to the host computer.
• The leaves of the tree are the I/O devices being served (for example, keyboard or speaker).
• 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.

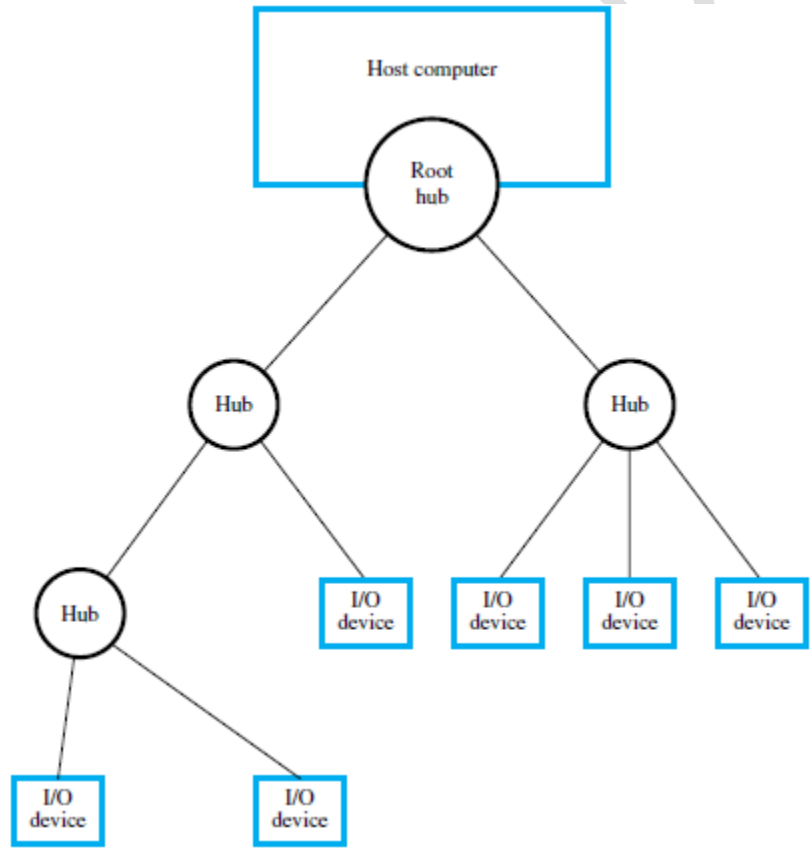


Figure 7.17 Universal Serial Bus tree structure.

USB ADDRESSING

• Each device may be a hub or an I/O device.
• Each device on the USB is assigned a 7-bit address.
• This address
→ is local to the USB tree and

→ is not related in any way to the addresses used on the processor-bus.
• A hub may have any number of devices or other hubs connected to it, and addresses are assigned arbitrarily.
• When a device is first connected to a hub, or when it is powered-on, it has the address 0.
• The hardware of the hub detects the device that has been connected, and it records this fact as part of its own status information.
• Periodically, the host polls each hub to
→ collect status information and
→ learn about new devices that may have been added or disconnected.
• When the host is informed that a new device has been connected, it uses sequence of commands to
→ send a reset signal on the corresponding hub port.
→ read information from the device about its capabilities.
→ send configuration information to the device, and
→ assign the device a unique USB address.
• Once this sequence is completed, the device
→ begins normal operation and
→ responds only to the new address.

USB PROTOCOLS

• All information transferred over the USB is organized in packets.
• A packet consists of one or more bytes of information.
• There are many types of packets that perform a variety of control functions.
• The information transferred on USB is divided into 2 broad categories: 1) Control and 2) Data.
• Control packets perform tasks such 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.
• A packet consists of one or more fields containing different kinds of information.
• The first field of any packet is called the **Packet Identifier (PID)** which identifies type of that packet.
• They are transmitted twice.
1) The first time they are sent with their true values and
2) The second time with each bit complemented.
• The four PID bits identify one of 16 different packet types.
• Some control packets, such as ACK (Acknowledge), consist only of the PID byte.
• Control packets used for controlling data transfer operations are called **Token Packets.**

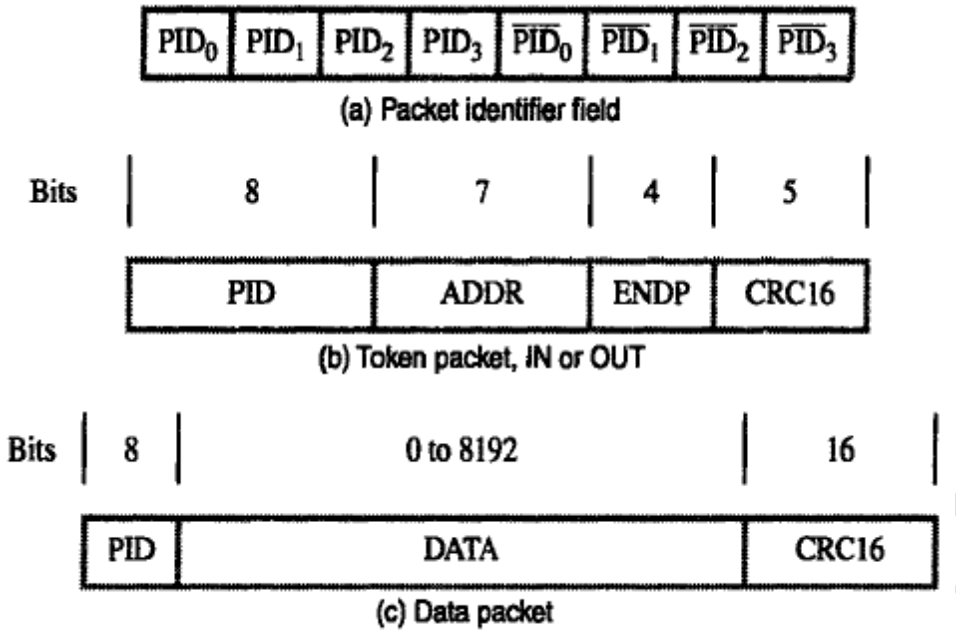


Figure 4.45 USB packet formats.

Problem 1:
The input status bit in an interface-circuit is cleared as soon as the input data register is read. Why is this important?

Solution:
After reading the input data, it is necessary to clear the input status flag before the program begins a new read-operation. Otherwise, the same input data would be read a second time.

Problem 2:
What is the difference between a subroutine and an interrupt-service routine?

Solution:
A subroutine is called by a program instruction to perform a function needed by the calling program. An interrupt-service routine is initiated by an event such as an input operation or a hardware error. The function it performs may not be at all related to the program being executed at the time of interruption. Hence, it must not affect any of the data or status information relating to that program.

Problem 3:
Three devices A, B, & C are connected to the bus of a computer. I/O transfers for all 3 devices use
interrupt control. Interrupt nesting for devices A & B is not allowed, but interrupt-requests from C may be accepted while either A or B is being serviced. Suggest different ways in which this can be accomplished in each of the following cases:

(a) The computer has one interrupt-request line.
(b) Two interrupt-request lines INTR1 & INTR2 are available, with INTR1 having higher priority. Specify when and how interrupts are enabled and disabled in each case.

Solution:

(a) Interrupts should be enabled, except when C is being serviced. The nesting rules can be enforced by manipulating the interrupt-enable flags in the interfaces of A and B.
(b) A and B should be connected to INTR , and C to INTR. When an interrupt-request is received from either A or B, interrupts from the other device will be automatically disabled until
the request has been serviced. However, interrupt-requests from C will always be accepted.

Problem 4:
Consider a computer in which several devices are connected to a common interrupt-request line. Explain how you would arrange for interrupts from device j to be accepted before the execution of the interrupt service routine for device i is completed. Comment in particular on the times at which interrupts must be enabled and disabled at various points in the system.

Solution:
Interrupts are disabled before the interrupt-service routine is entered. Once device i turns off its interrupt-request, interrupts may be safely enabled in the processor. If the interface-circuit of device i turns off its interrupt-request when it receives the interrupt acknowledge signal, interrupts may be enabled at the beginning of the interrupt-service routine of device i. Otherwise, interrupts may be enabled only after the instruction that causes device i to turn off its interrupt-request has been executed.

Problem 5:
Consider the daisy chain arrangement. Assume that after a device generates an interrupt-request, it turns off that request as soon as it receives the interrupt acknowledge signal. Is it still necessary to disable interrupts in the processor before entering the interrupt service routine? Why?

Solution:
Yes, because other devices may keep the interrupt-request line asserted.