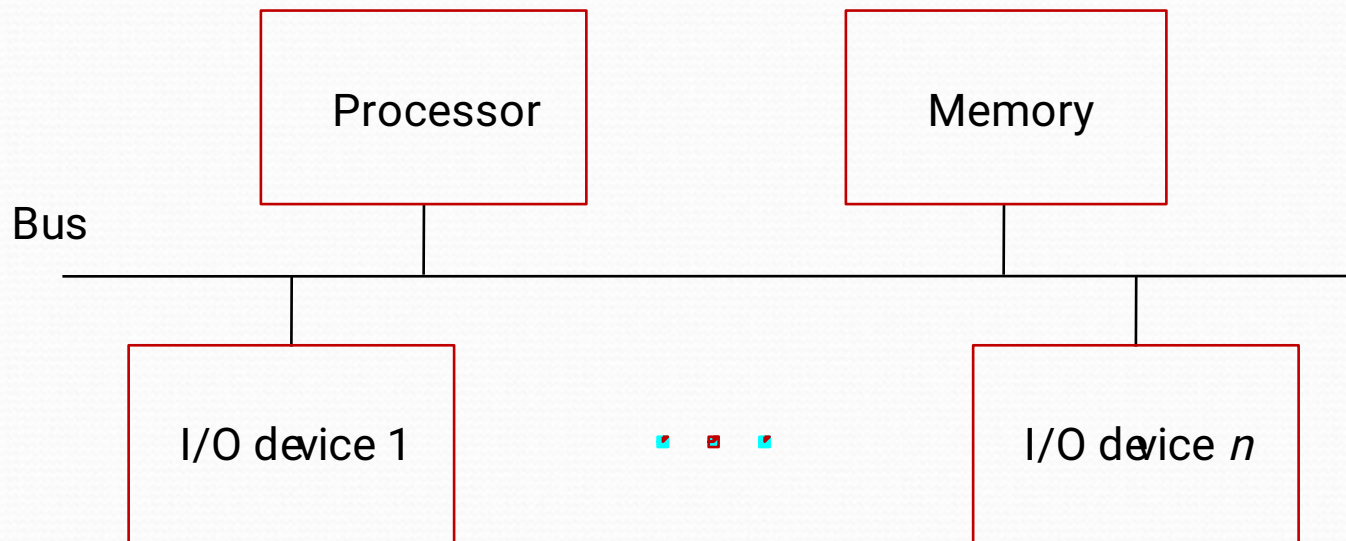# INPUT/OUTPUT ORGANIZATION

Accessing I/O Devices

Accessing I/O devices

- Multiple I/O devices may be connected to the processor and the memory via a bus.
- Bus consists of three sets of lines to carry address, data and control signals.
- Each I/O device is assigned an unique address.
- To access an I/O device, the processor places the address on the address lines.
- The device recognizes the address, and responds to the control signals.

Accessing I/O devices (contd. .)

- I/O devices and the memory may share the same address space:
  - Memory-mapped I/O.
  - Any machine instruction that can access memory can be used to transfer data to or from an I/O device.
  - Simpler software.
- I/O devices and the memory may have different address spaces:
  - Special instructions to transfer data to and from I/O devices.
  - I/O devices may have to deal with fewer address lines.
  - I/O address lines need not be physically separate from memory address lines.
  - In fact, address lines may be shared between I/O devices and memory, with a control signal to indicate whether it is a memory address or an I/O address.

4

Accessing I/O devices (contd..)
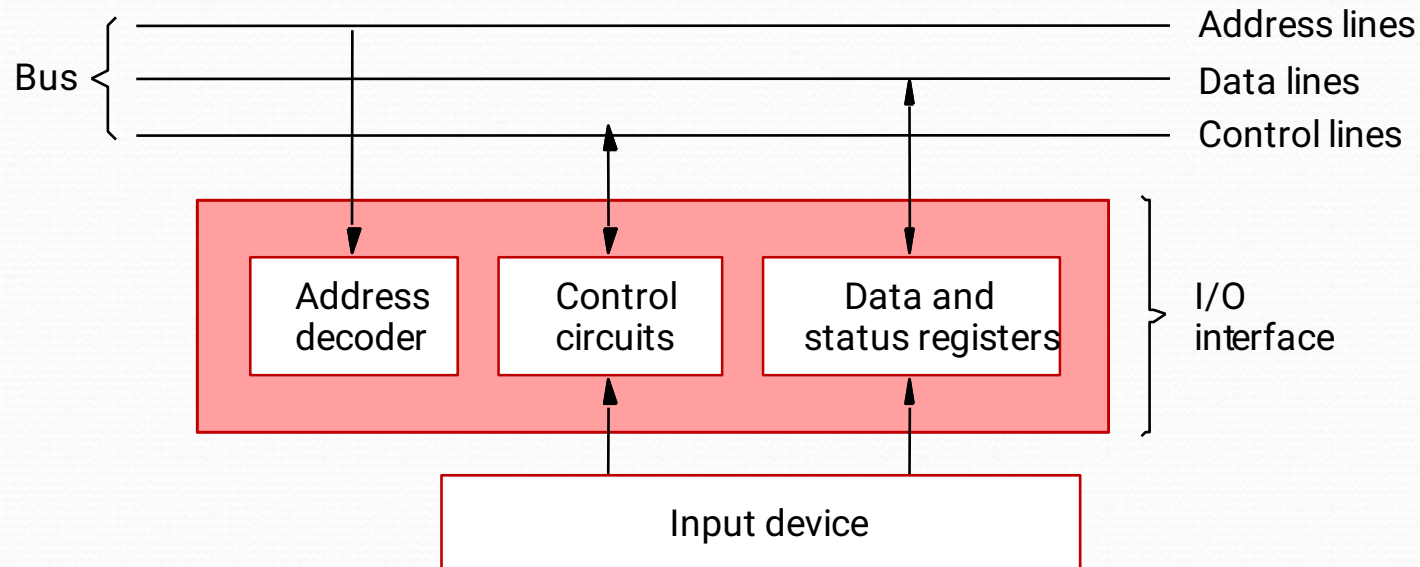
- I/O device is connected to the bus using an I/O interface circuit which has:
  - Address decoder, control circuit, and data and status registers.
- Address decoder decodes the address placed on the address lines thus enabling the device to recognize its address.
- Data register holds the data being transferred to or from the processor.
- Status register holds information necessary for the operation of the I/O device.
- Data and status registers are connected to the data lines, and have unique addresses.
- I/O interface circuit coordinates I/O transfers.

Cntd..
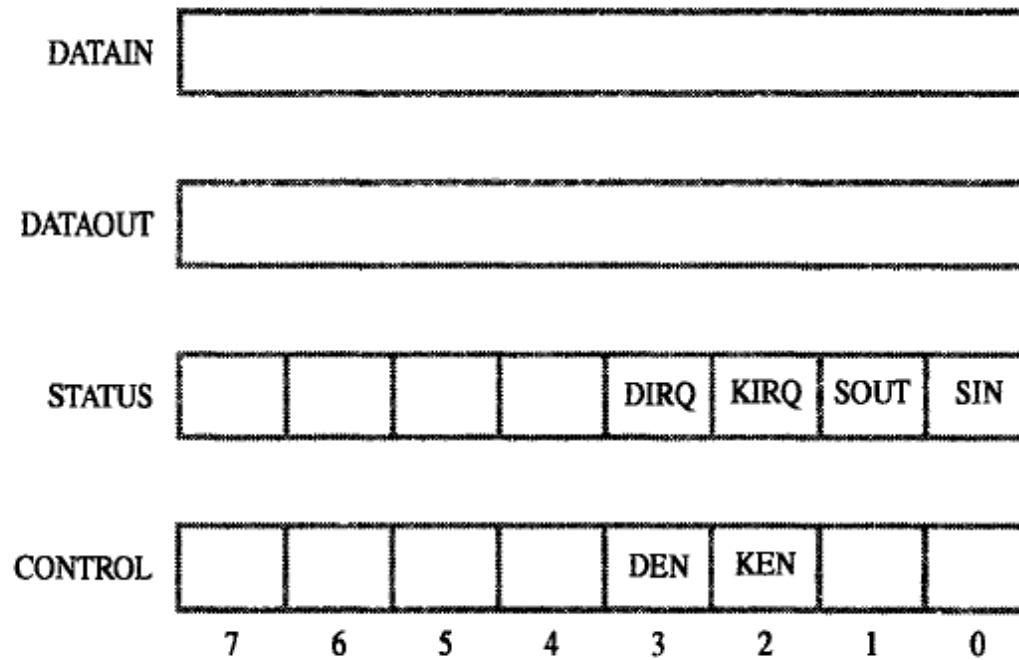
Figure 4.3 Registers in keyboard and display interfaces.

```
        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.

Accessing I/O devices (contd.)

- Recall that the rate of transfer to and from I/O devices is slower than the speed of the processor. This creates the need for mechanisms to synchronize data transfers between them.
- Program-controlled I/O:
  - Processor repeatedly monitors a status flag to achieve the necessary synchronization.
  - Processor polls the I/O device.
- Two other mechanisms used for synchronizing data transfers between the processor and memory:
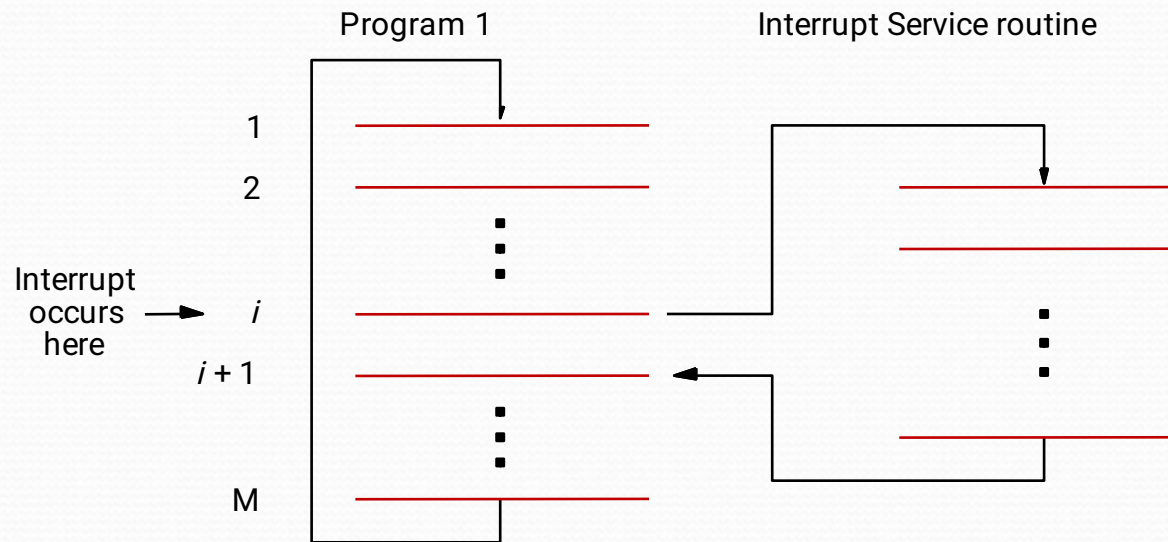 - Interrupts.
 - Direct Memory Access.

Interrupts

Interrupts

- In program-controlled I/O, when the processor continuously monitors the status of the device, it does not perform any useful tasks.

- An alternate approach would be for the I/O device to alert the processor when it becomes ready.
  - Do so by sending a hardware signal called an interrupt to the processor.
  - At least one of the bus control lines, called an interrupt-request line is dedicated for this purpose.

- Processor can perform other useful tasks while it is waiting for the device to be ready.

Interrupts (contd..)

Program 1

Interrupt Service routine

1
2
i
i + 1
M

Interrupt
occurs
here

- Processor is executing the instruction located at address i when an interrupt occurs.
- Routine executed in response to an interrupt request is called the interrupt-service routine.
- When an interrupt occurs, control must be transferred to the interrupt service routine.
- But before transferring control, the current contents of the PC (i+1), must be saved in a known location.
- This will enable the return-from-interrupt instruction to resume execution at i+1.
- Return address, or the contents of the PC are usually stored on the processor stack.

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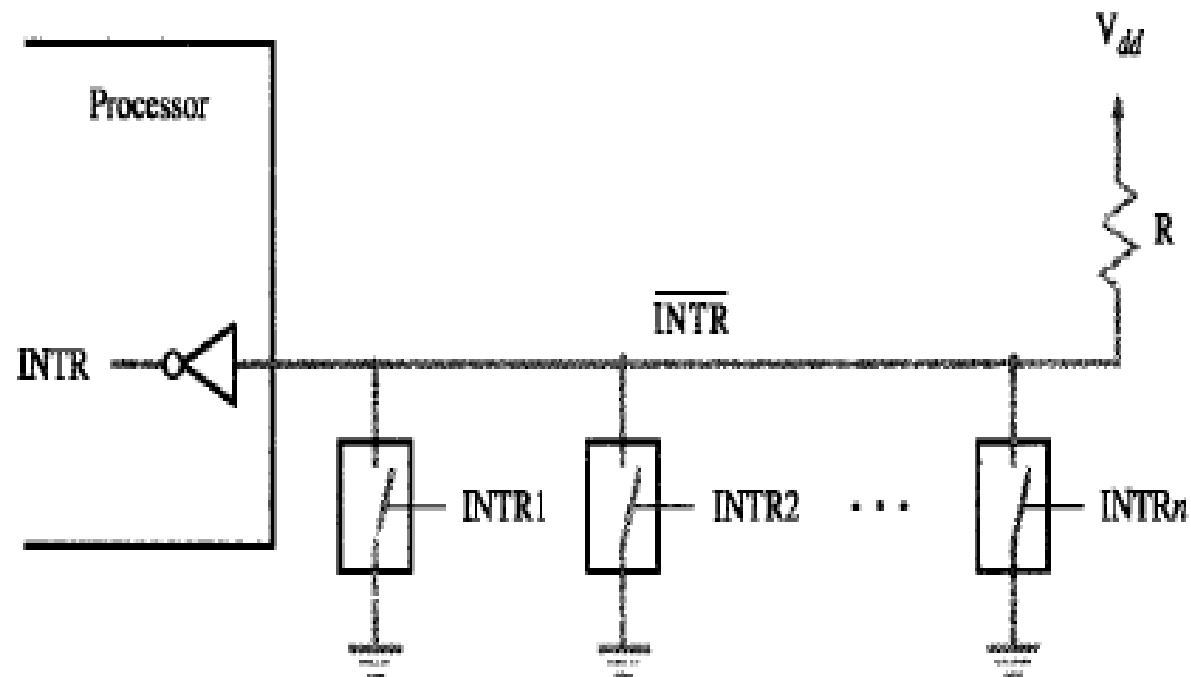
Figure 4.6 An equivalent circuit for an open-drain bus used to implement a common interrupt-request line.

Interrupts (contd..)

- Treatment of an interrupt-service routine is very similar to that of a subroutine.
- However there are significant differences:
  - A subroutine performs a task that is required by the calling program.
  - Interrupt-service routine may not have anything in common with the program it interrupts.
  - Interrupt-service routine and the program that it interrupts may belong to different users.
  - As a result, before branching to the interrupt-service routine, not only the PC, but other information such as condition code flags, and processor registers used by both the interrupted program and the interrupt service routine must be stored.
  - This will enable the interrupted program to resume execution upon return from interrupt service routine.

Enabling and disabling of Interrupt

- Saving and restoring information can be done automatically by the processor or explicitly by program instructions.

- Saving and restoring registers involves memory transfers:
  - Increases the total execution time.
  - Increases the delay between the time an interrupt request is received, and the start of execution of the interrupt-service routine. This delay is called interrupt latency.

- In order to reduce the interrupt latency, most processors save only the minimal amount of information:
  - This minimal amount of information includes Program Counter and processor status registers.

- Any additional information that must be saved, must be saved explicitly by the program instructions at the beginning of the interrupt service routine.

Interrupts (contd..)

- When a processor receives an interrupt-request, it must branch to the interrupt service routine.

- It must also inform the device that it has recognized the interrupt request.

- This can be accomplished in two ways:
  - Some processors have an explicit interrupt-acknowledge control signal for this purpose.
  - In other cases, the data transfer that takes place between the device and the processor can be used to inform the device.

Interrupts (contd..)

- Interrupt-requests interrupt the execution of a program, and may alter the intended sequence of events:
  - Sometimes such alterations may be undesirable, and must not be allowed.
  - For example, the processor may not want to be interrupted by the same device while executing its interrupt-service routine.
- Processors generally provide the ability to enable and disable such interruptions as desired.
- One simple way is to provide machine instructions such as *Interrupt-enable* and *Interrupt-disable* for this purpose.
- To avoid interruption by the same device during the execution of an interrupt service routine:
  - First instruction of an interrupt service routine can be Interrupt-disable.
  - Last instruction of an interrupt service routine can be Interrupt-enable.

Handling Multiple Devices

- Multiple I/O devices may be connected to the processor and the memory via a bus. Some or all of these devices may be capable of generating interrupt requests.
  - Each device operates independently, and hence no definite order can be imposed on how the devices generate interrupt requests?
- How does the processor know which device has generated an interrupt?
- How does the processor know which interrupt service routine needs to be executed?
- When the processor is executing an interrupt service routine for one device, can other device interrupt the processor?
- If two interrupt-requests are received simultaneously,

contd..

- Consider a simple arrangement where all devices send their interrupt-requests over a single control line in the bus.

- When the processor receives an interrupt request over this control line, how does it know which device is requesting an interrupt?

- This information is available in the status register of the device requesting an interrupt:
  - The status register of each device has an *IRQ* bit which it sets to 1 when it requests an interrupt.

- Interrupt service routine can poll the I/O devices connected to the bus. The first device with *IRQ* equal to 1 is the one that is serviced.

- Polling mechanism is easy, but time consuming to query the status bits of all the I/O devices connected to the bus.

1.Vectored Interrupt

- The device requesting an interrupt may identify itself directly to the processor.
  - Device can do so by sending a special code (4 to 8 bits) the processor over the bus.
  - Code supplied by the device may represent a part of the starting address of the interrupt-service routine.
  - The remainder of the starting address is obtained by the processor based on other information such as the range of memory addresses where interrupt service routines are located.
- Usually the location pointed to by the interrupting device is used to store the starting address of the interrupt-service routine.

contd..

- Previously, before the processor started executing the interrupt service routine for a device, it disabled the interrupts from the device.

- In general, same arrangement is used when multiple devices can send interrupt requests to the processor.

  - During the execution of an interrupt service routine of device, the processor does not accept interrupt requests from any other device.
  - Since the interrupt service routines are usually short, the delay that this causes is generally acceptable.

- However, for certain devices this delay may not be acceptable.

  - Which devices can be allowed to interrupt a processor when it is executing an interrupt service routine of another device?

2.Interrupt Nesting

- I/O devices are organized in a priority structure:
  - An interrupt request from a high-priority device is accepted while the processor is executing the interrupt service routine of a low priority device.

- A priority level is assigned to a processor that can be changed under program control.
  - Priority level of a processor is the priority of the program that is currently being executed.
  - When the processor starts executing the interrupt service routine of a device, its priority is raised to that of the device.
  - If the device sending an interrupt request has a higher priority than the processor, the processor accepts the interrupt request.

contd..

- Processor's priority is encoded in a few bits of the processor status register.
    - Priority can be changed by instructions that write into the processor status register.
    - Usually, these are privileged instructions, or instructions that can be executed only in the supervisor mode.
    - Privileged instructions cannot be executed in the user mode.
    - Prevents a user program from accidentally or intentionally changing the priority of the processor.
- If there is an attempt to execute a privileged instruction in the user mode, it causes a special type of interrupt called as privilege exception.

Interrupt Nesting contd..
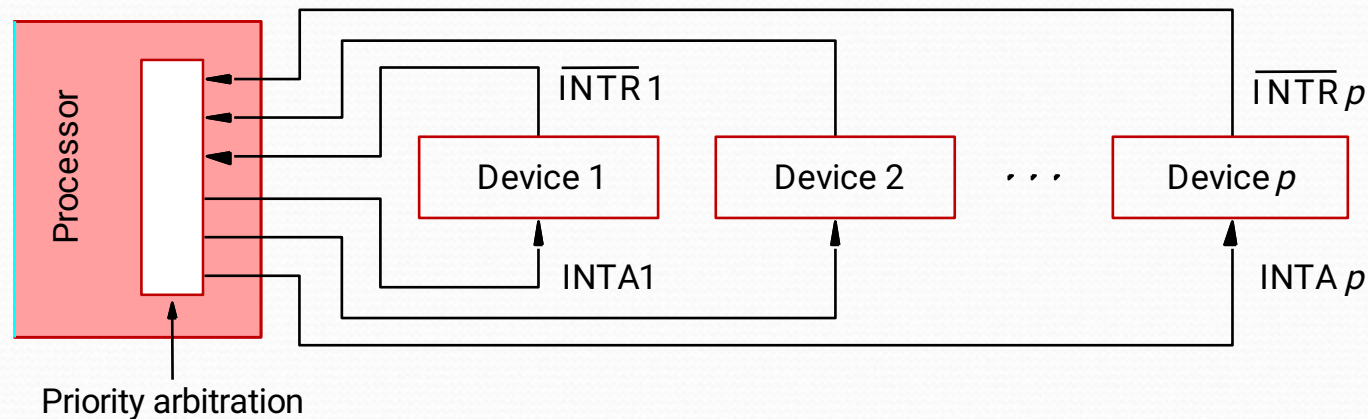
- Each device has a separate interrupt-request and interrupt-acknowledge line.
- Each interrupt-request line is assigned a different priority level.
- Interrupt requests received over these lines are sent to a priority arbitration circuit in the processor.
- If the interrupt request has a higher priority level than the priority of the processor, then the request is accepted.

cntd..

- Which interrupt request does the processor accept if it receives interrupt requests from two or more devices simultaneously?.

- If the I/O devices are organized in a priority structure, the processor accepts the interrupt request from a device with higher priority.

  - Each device has its own interrupt request and interrupt acknowledge line.
  - A different priority level is assigned to the interrupt request line of each device.

- However, if the devices share an interrupt request line, then how does the processor decide which interrupt request to accept?

Simultaneous Request

Polling scheme:
- *If the processor uses a polling mechanism to poll the status registers of I/O devices to determine which device is requesting an interrupt.*
- *In this case the priority is determined by the order in which the devices are polled.*
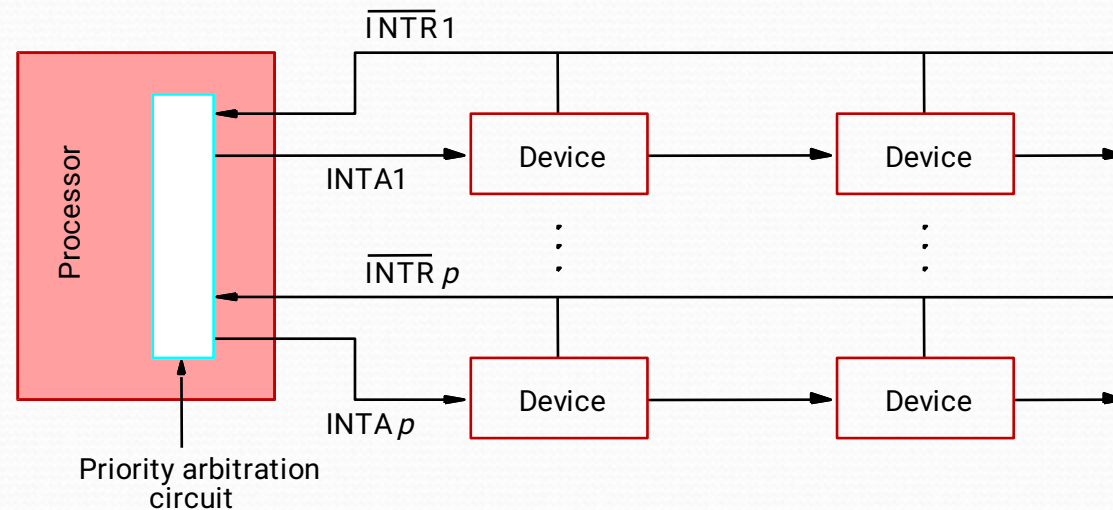- *The first device with status bit set to 1 is the device whose interrupt request is accepted.*

Daisy chain scheme:
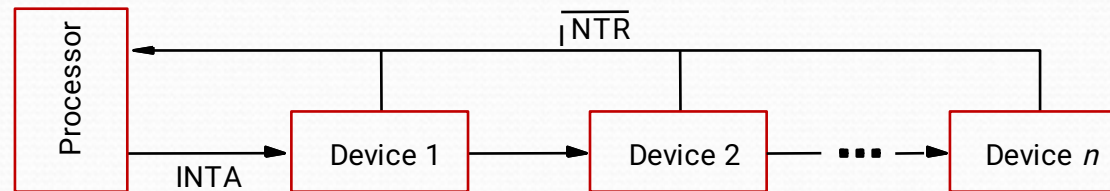
- *Devices are connected to form a daisy chain.*
- *Devices share the interrupt-request line, and interrupt-acknowledge line is connected to form a daisy chain.*
- *When devices raise an interrupt request, the interrupt-request line is activated.*
- *The processor in response activates interrupt-acknowledge.*
- *Received by device 1, if device 1 does not need service, it passes the signal to device 2.*
- *Device that is electrically closest to the processor has the highest priority.*

Interrupts (contd..)

- When I/O devices were organized into a priority structure, each device had its own interrupt-request and interrupt-acknowledge line.
- When I/O devices were organized in a daisy chain fashion, the devices shared an interrupt-request line, and the interrupt-acknowledge propagated through the devices.
- A combination of priority structure and daisy chain scheme can also used.

$$\overline{INTR1}$$

Processor

INTA1

$$\overline{INTR}\,p$$

INTA p

Priority arbitration circuit

Device   Device

Device   Device

- Devices are organized into groups.
- Each group is assigned a different priority level.
- All the devices within a single group share an interrupt-request line, and are connected to form a daisy chain.

Controlling Device Request

- Only those devices that are being used in a program should be allowed to generate interrupt requests.
- To control which devices are allowed to generate interrupt requests, the interface circuit of each I/O device has an interrupt-enable bit.
    - If the interrupt-enable bit in the device interface is set to 1, then the device is allowed to generate an interrupt-request.
- Interrupt-enable bit in the device's interface circuit determines whether the device is allowed to generate an interrupt request.
- Interrupt-enable bit in the processor status register or the priority structure of the interrupts determines whether a given interrupt will be accepted.

Cntd..

1. Load the starting address of the interrupt-service routine in location INTVEC.

2. Load the address LINE in a memory location PNTR. The interrupt-service routine will use this location as a pointer to store the input characters in the memory.

3. Enable keyboard interrupts by setting bit 2 in register CONTROL to 1.

4. Enable interrupts in the processor by setting to 1 the IE bit in the processor status register, PS.

Once this initialization is completed, typing a character on the keyboard will cause an interrupt request to be generated by the keyboard interface. The program being executed at that time will be interrupted and the interrupt-service routine will be executed. This routine has to perform the following tasks:

1. Read the input character from the keyboard input data register. This will cause the interface circuit to remove its interrupt request.

2. Store the character in the memory location pointed to by PNTR, and increment PNTR.

3. When the end of the line is reached, disable keyboard interrupts and inform program Main.

4. Return from interrupt.

Cntd..

Main Program

	Move	#LINE,PNTR	Initialize buffer pointer.
	Clear	EOL	Clear end-of-line indicator.
	BitSet	#2,CONTROL	Enable keyboard interrupts.
	BitSet	#9,PS	Set interrupt-enable bit in the PS.

$\vdots$

Interrupt-service routine

READ	MoveMultiple	R0–R1,−(SP)	Save registers R0 and R1 on stack.
	Move	PNTR,R0	Load address pointer.
	MoveByte	DATAIN,R1	Get input character and
	MoveByte	R1,(R0)+	store it in memory.
	Move	R0,PNTR	Update pointer.
	CompareByte	#$0D,R1	Check if Carriage Return.
	Branch≠0	RTRN	
	Move	#1,EOL	Indicate end of line.
	BitClear	#2,CONTROL	Disable keyboard interrupts.
RTRN	MoveMultiple	(SP)+,R0–R1	Restore registers R0 and R1.
	Return-from-interrupt		

Figure 4.9 Using interrupts to read a line of characters from a keyboard via the registers in Figure 4.3.

Exceptions

- Interrupts caused by interrupt-requests sent by I/O devices.

- Interrupts could be used in many other situations where the execution of one program needs to be suspended and execution of another program needs to be started.

- In general, the term exception is used to refer to any event that causes an interruption.

  - Interrupt-requests from I/O devices is one type of an exception.

- Other types of exceptions are:

  - Recovery from errors
  - Debugging
  - Privilege exception

Exceptions (contd..)

- Many sources of errors in a processor. For example:
  - Error in the data stored.
  - Error during the execution of an instruction.
- When such errors are detected, exception processing is initiated.
  - Processor takes the same steps as in the case of I/O interrupt-request.
  - It suspends the execution of the current program, and starts executing an exception-service routine.
- Difference between handling I/O interrupt-request and handling exceptions due to errors:
  - In case of I/O interrupt-request, the processor usually completes the execution of an instruction in progress before branching to the interrupt-service routine.

Exceptions (contd..)

- Debugger uses exceptions to provide important features:
  - Trace,
  - Breakpoints.

- Trace mode:
  - Exception occurs after the execution of every instruction.
  - Debugging program is used as the exception-service routine.

- Breakpoints:
  - Exception occurs only at specific points selected by the user.
  - Debugging program is used as the exception-service routine.

Exceptions (contd..)

- Certain instructions can be executed only when the processor is in the supervisor mode. These are called privileged instructions.

- If an attempt is made to execute a privileged instruction in the user mode, a privilege exception occurs.

- Privilege exception causes:
  - Processor to switch to the supervisor mode,
  - Execution of an appropriate exception-servicing routine.

ACCESSING I/O DEVICES

A simple arrangement to connect I/O devices to a computer is to use a single bus arrangement. The bus enables all the devices connected to it to exchange information. Typically, it consists of three sets of lines used to carry address, data, and control signals. Each I/O device is assigned a unique set of addresses. When the processor places a particular address on the address line, the device that recognizes this address responds to the commands issued on the control lines. The processor requests either a read or a write operation, and the requested data are transferred over the data lines, when I/O devices and the memory share the same address space, the arrangement is called memory-mapped I/O.

With memory-mapped I/O, any machine instruction that can access memory can be used to transfer data to or from an I/O device. For example, if DATAIN is the address of the input buffer associated with the keyboard, the instruction

Move DATAIN, R0

Reads the data from DATAIN and stores them into processor register R0. Similarly, the instruction

Move R0, DATAOUT

Sends the contents of register R0 to location DATAOUT, which may be the output data buffer of a display unit or a printer.

Most computer systems use memory-mapped I/O. some processors have special In and Out instructions to perform I/O transfers. When building a computer system based on these processors, the designer had the option of connecting I/O devices to use the special I/O address space or simply incorporating them as part of the memory address space. The I/O devices examine the low-order bits of the address bus to determine whether they should respond.

The hardware required to connect an I/O device to the bus. The address decoder enables the device to recognize its address when this address appears on the address lines. The data register holds the data being transferred to or from the processor. The status register contains information relevant to the operation of the I/O device. Both the data and status registers are connected to the data bus and assigned unique addresses. The address decoder, the data and status registers, and the control circuitry required to coordinate I/O transfers constitute the device's interface circuit.

I/O devices operate at speeds that are vastly different from that of the processor. When a human operator is entering characters at a keyboard, the processor is capable of executing millions of instructions between successive character entries. An instruction that reads a character from the keyboard should be executed only when a character is available in the input buffer of the keyboard interface. Also, we must make sure that an input character is read only once.

This example illustrates program-controlled I/O, in which the processor repeatedly checks a status flag to achieve the required synchronization between the processor and an input or output device. We say that the processor polls the device. There are two other commonly used mechanisms for implementing I/O operations: interrupts and direct memory access. In the case of interrupts, synchronization is achieved by having the I/O device send a special signal over the bus whenever it is ready for a data transfer operation. Direct memory access is a technique used for high-speed I/O devices. It involves having the device interface transfer data directly to or from the memory, without continuous involvement by the processor.

The routine executed in response to an interrupt request is called the interrupt-service routine, which is the PRINT routine in our example. Interrupts bear considerable resemblance to subroutine calls. Assume that an interrupt request arrives during execution of instruction i in figure 1

Program 1
COMPUTER routine

Program 2
PRINT routine
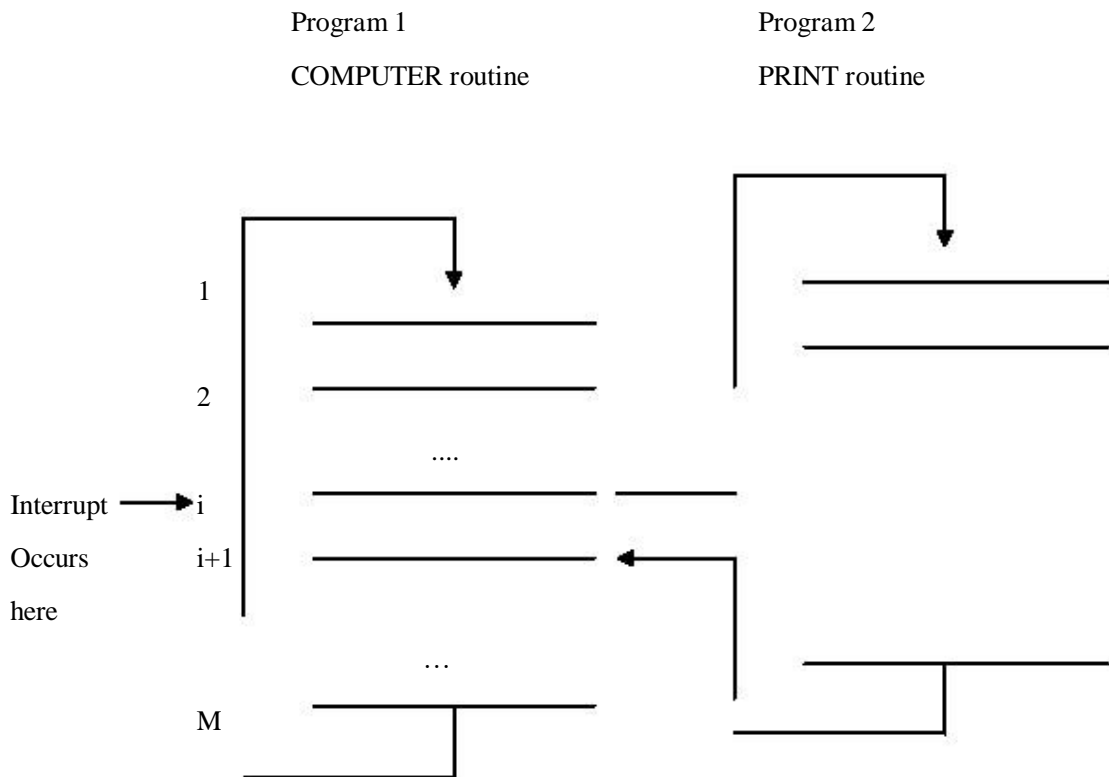
1

2

....

Interrupt → i

Occurs   i+1

here

…

M

Figure 1. Transfer of control through the use of interrupts

The processor first completes execution of instruction i. Then, it loads the program counter with the address of the first instruction of the interrupt-service routine. For the time being, let us assume that this address is hardwired in the processor. After execution of the interrupt-service routine, the processor has to come back to instruction i +1. Therefore, when an interrupt occurs, the current contents of the PC, which point to instruction i+1, must be put in temporary storage in a known location. A Return-from-interrupt instruction at the end of the interrupt-service routine reloads the PC from the temporary storage location, causing execution to resume at instruction i +1. In many processors, the return address is saved on the processor stack.

We should note that as part of handling interrupts, the processor must inform the device that its request has been recognized so that it may remove its interrupt-request

signal. This may be accomplished by means of a special control signal on the bus. An interrupt-acknowledge signal. The execution of an instruction in the interrupt-service routine that accesses a status or data register in the device interface implicitly informs that device that its interrupt request has been recognized.

So far, treatment of an interrupt-service routine is very similar to that of a subroutine. An important departure from this similarity should be noted. A subroutine performs a function required by the program from which it is called. However, the interrupt-service routine may not have anything in common with the program being executed at the time the interrupt request is received. In fact, the two programs often belong to different users. Therefore, before starting execution of the interrupt-service routine, any information that may be altered during the execution of that routine must be saved. This information must be restored before execution of the interrupt program is resumed. In this way, the original program can continue execution without being affected in any way by the interruption, except for the time delay. The information that needs to be saved and restored typically includes the condition code flags and the contents of any registers used by both the interrupted program and the interrupt-service routine.

The task of saving and restoring information can be done automatically by the processor or by program instructions. Most modern processors save only the minimum amount of information needed to maintain the registers involves memory transfers that increase the total execution time, and hence represent execution overhead. Saving registers also increase the delay between the time an interrupt request is received and the start of execution of the interrupt-service routine. This delay is called interrupt latency.

3.2 INTERRUPT HARDWARE:

We pointed out that an I/O device requests an interrupt by activating a bus line called interrupt-request. Most computers are likely to have several I/O devices that can request an interrupt. A single interrupt-request line may be used to serve n devices as depicted. All devices are connected to the line via switches to ground. To request an interrupt, a device closes its associated switch. Thus, if all interrupt-request signals $INTR_1$ to $INTR_n$ are inactive, that is, if all switches are open, the voltage on the interrupt-request line will be equal to V_{dd}. This is the inactive state of the line. Since the closing of

one or more switches will cause the line voltage to drop to 0, the value of INTR is the

logical OR of the requests from individual devices, that is,

$$INTR = INTR_1 + \ldots\ldots + INTR_n$$

It is customary to use the complemented form, *INTR* , to name the interrupt-request

signal on the common line, because this signal is active when in the low-voltage state.

3.3 ENABLING AND 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 the envisaged by the programmer. Hence, the

interruption of program execution must be carefully controlled.

Let us consider in detail the specific case of a single interrupt request from one

device. When a device activates the interrupt-request signal, it keeps this signal activated

until it learns that the processor has accepted its request. This means that the interrupt-

request signal will be active during execution of the interrupt-service routine, perhaps

until an instruction is reached that accesses the device in question.

The first possibility is to have the processor hardware ignore the interrupt-request

line until the execution of the first instruction of the interrupt-service routine has been

completed. Then, by using an Interrupt-disable instruction as the first instruction in the

interrupt-service routine, the programmer can ensure that no further interruptions will

occur until an Interrupt-enable instruction is executed. Typically, the Interrupt-enable

instruction will be the last instruction in the interrupt-service routine before the Return-

from-interrupt instruction. The processor must guarantee that execution of the Return-

from-interrupt instruction is completed before further interruption can occur.

The second option, which is suitable for a simple processor with only one

interrupt-request line, is to have the processor automatically disable interrupts before

starting the execution of the interrupt-service routine. After saving the contents of the PC

and the processor status register (PS) on the stack, the processor performs the equivalent of executing an Interrupt-disable instruction. It is often the case that one bit in the PS register, called Interrupt-enable, indicates whether interrupts are enabled.

In the third option, the processor has a special interrupt-request line for which the interrupt-handling circuit responds only to the leading edge of the signal. Such a line is said to be edge-triggered.

Before proceeding to study more complex aspects of interrupts, let us summarize the sequence of events involved in handling an interrupt request from a single device. Assuming that interrupts are enabled, the following is a typical scenario.

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 PS (except in the case of edge-triggered interrupts).
4. The device is informed that its request has been recognized, and in response, it 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.

3.4 HANDLING MULTIPLE DEVICES:

Let us now consider the situation where a number of devices capable of initiating interrupts are connected to the processor. Because these devices are operationally independent, there is no definite order in which they will generate interrupts. For example, device X may request in interrupt while an interrupt caused by device Y is being serviced, or several devices may request interrupts at exactly the same time. This gives rise to a number of questions

1. How can the processor recognize the device requesting an interrupts?
2. Given that different devices are likely to require different interrupt-service routines, how can the processor obtain the starting address of the appropriate routine in each case?

3. Should a device be allowed to interrupt the processor while another interrupt is being serviced?

4. How should two or more simultaneous interrupt requests be handled?

The means by which these problems are resolved vary from one computer to another, And the approach taken is an important consideration in determining the computer's suitability for a given application.

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.

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.

Vectored Interrupts:-

To reduce the time involved in the polling process, a device requesting an interrupt may identify itself directly to the processor. Then, the processor can immediately start executing the corresponding interrupt-service routine. The term vectored interrupts refers to all interrupt-handling schemes based on this approach.

A device requesting an interrupt can identify itself by sending a special code to the processor over the bus. This enables the processor to identify individual devices even if they share a single interrupt-request line. The code supplied by the device may represent the starting address of the interrupt-service routine for that device. The code

length is typically in the range of 4 to 8 bits. The remainder of the address is supplied by the processor based on the area in its memory where the addresses for interrupt-service routines are located.

This arrangement implies that the interrupt-service routine for a given device must always start at the same location. The programmer can gain some flexibility by storing in this location an instruction that causes a branch to the appropriate routine.

Interrupt Nesting: -

Interrupts should be disabled during the execution of an interrupt-service routine, to ensure that a request from one device will not cause more than one interruption. The same arrangement is often used when several devices are involved, in which case execution of a given interrupt-service routine, once started, always continues to completion before the processor accepts an interrupt request from a second device. Interrupt-service routines are typically short, and the delay they may cause is acceptable for most simple devices.

For some devices, however, a long delay in responding to an interrupt request may lead to erroneous operation. Consider, for example, a computer that keeps track of the time of day using a real-time clock. This is a device that sends interrupt requests to the processor at regular intervals. For each of these requests, the processor executes a short interrupt-service routine to increment a set of counters in the memory that keep track of time in seconds, minutes, and so on. Proper operation requires that the delay in responding to an interrupt request from the real-time clock be small in comparison with the interval between two successive requests. To ensure that this requirement is satisfied in the presence of other interrupting devices, it may be necessary to accept an interrupt request from the clock during the execution of an interrupt-service routine for another device.

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.
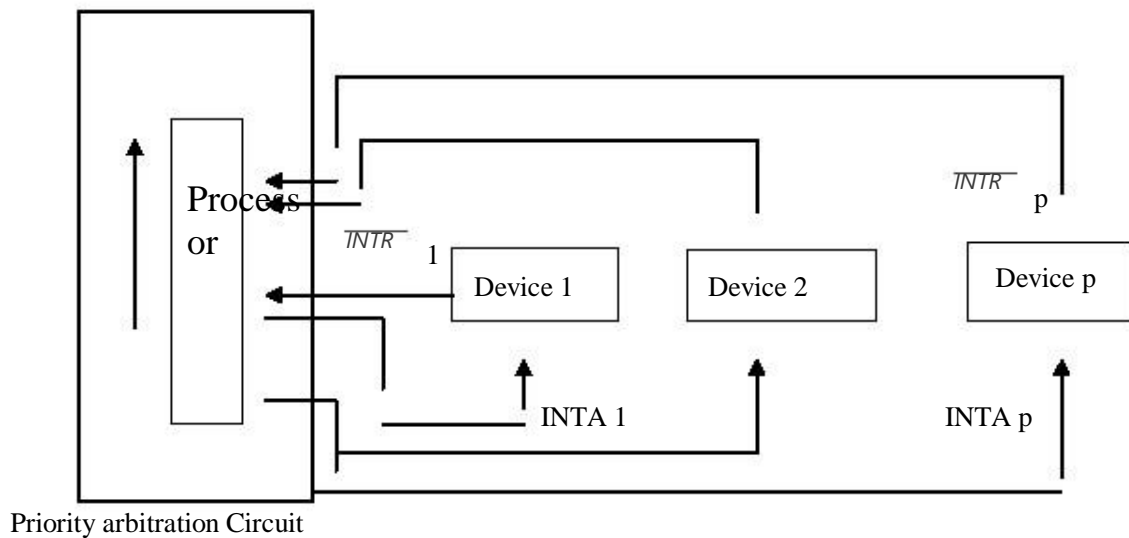
Priority arbitration Circuit

Figure2: Implementation of interrupt priority using individual interrupt-request and acknowledge lines.

Simultaneous Requests:-

Let us now consider the problem of simultaneous arrivals of interrupt requests from two or more devices. The processor must have some means of deciding which requests to service first. Using a priority scheme such as that of figure, the solution is straightforward. The processor simply accepts the requests having the highest priority.

Polling the status registers of the I/O devices is the simplest such mechanism. In this case, priority is determined by the order in which the devices are polled. When vectored interrupts are used, we must ensure that only one device is selected to send its interrupt vector code. A widely used scheme is to connect the devices to form a daisy chain, as shown in figure 3a. The interrupt-request line $\overline{INTR}$ is common to all devices.

The interrupt-acknowledge line, INTA, is connected in a daisy-chain fashion, such that the INTA signal propagates serially through the devices.
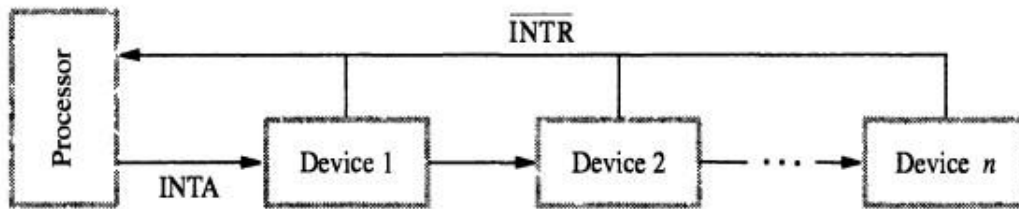
(a) Daisy chain
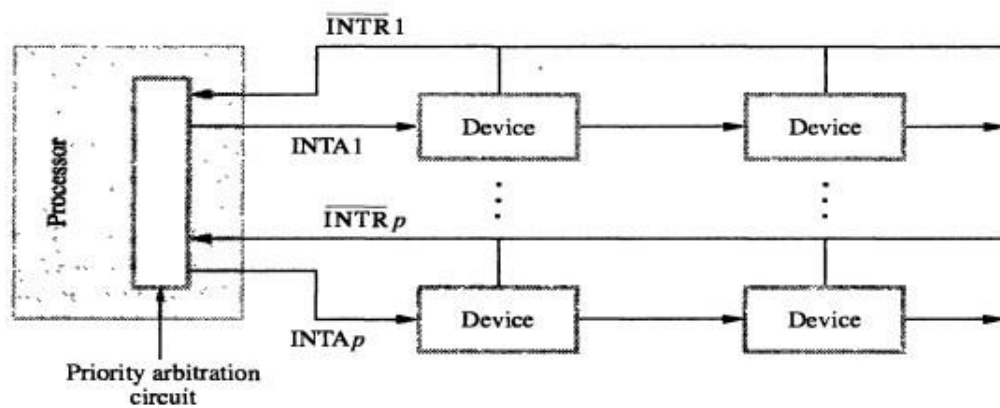
(b) Arrangement of priority groups

Figure 4.8   Interrupt priority schemes.

When several devices raise an interrupt request and the *INTR* line is activated,

the processor responds by setting the INTA line to 1. This signal is received by device 1.

Device 1 passes the signal on to device 2 only if it does not require any service. If device 1 has a pending request for interrupt, it blocks the INTA signal and proceeds to put its identifying code on the data lines. Therefore, in the daisy-chain arrangement, the device that is electrically closest to the processor has the highest priority. The second device along the chain has second highest priority, and so on.

The scheme in figure 3.a requires considerably fewer wires than the individual connections in figure 2. The main advantage of the scheme in figure 2 is that it allows the processor to accept interrupt requests from some devices but not from others, depending upon their priorities. The two schemes may be combined to produce the more general structure in figure 3b. Devices are organized in groups, and each group is connected at a different priority level. Within a group, devices are connected in a daisy chain. This organization is used in many computer systems.

3.5 CONTROLLING DEVICE REQUESTS:

Until now, we have assumed that an I/O device interface generates an interrupt request whenever it is ready for an I/O transfer, for example whenever the SIN flag is 1. It is important to ensure that interrupt requests are generated only by those I/O devices that are being used by a given program. Idle devices must not be allowed to generate interrupt requests, even though they may be ready to participate in I/O transfer operations. Hence, we need a mechanism in the interface circuits of individual devices to control whether a device is allowed to generate an interrupt request.

The control needed is usually provided in the form of an interrupt-enable bit in the device's interface circuit. The keyboard interrupt-enable, KEN, and display interrupt-enable, DEN, flags in register CONTROL perform this function. If either of these flags is set, the interface circuit generates an interrupt request whenever the corresponding status flag in register STATUS is set. At the same time, the interface circuit sets bit KIRQ or DIRQ to indicate that the keyboard or display unit, respectively, is requesting an

interrupt. If an interrupt-enable bit is equal to 0, the interface circuit will not generate an interrupt request, regardless of the state of the status flag.

To summarize, there are two independent mechanisms for controlling interrupt requests. At the device end, an interrupt-enable bit in a control register determines whether the device is allowed to generate an interrupt request. At the processor end, either an interrupt enable bit in the PS register or a priority structure determines whether a given interrupt request will be accepted.

3.6 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 requests received during 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. We now describe a few other kinds of exceptions.

Recovery from Errors:

Computers use a variety of techniques to ensure that all hardware components are operating properly. For example, many computers include an error-checking code in the main memory, which allows detection of errors in the stored data. If errors occur, the control hardware detects it and informs the processor by raising an interrupt.

The processor may also interrupt a program if it detects an error or an unusual condition while executing the instructions of this program. For example, the OP-code field of an instruction may not correspond to any legal instruction, or an arithmetic instruction may attempt a division by zero.

When exception processing is initiated as a result of such errors, the processor proceeds in exactly the same manner as in the case of an I/O interrupt request. It suspends the program being executed and starts an exception-service routine. This routine takes appropriate action to recover from the error, if possible, or to inform the user about it. Recall that in the case of an I/O interrupt, the processor completes execution of the instruction in progress before accepting the interrupt. However, when an interrupt is caused by an error, execution of the interrupted instruction cannot usually be completed, and the processor begins exception processing immediately.

Debugging:

Another important type of exception is used as an aid in debugging programs. System software usually includes a program called a debugger, which helps the programmer find errors in a program. The debugger uses exceptions to provide two important facilities called trace and breakpoints.

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

Breakpoint provides a similar facility, except that the program being debugged is interrupted only at specific points selected by the user. An instruction called Trap or Software-interrupt is usually provided for this purpose. Execution of this instruction results in exactly the same actions as when a hardware interrupt request is received. While debugging a program, the user may wish to interrupt program execution after instruction i. The debugging routine saves instruction i+1 and replaces it with a software interrupt instruction. When the program is executed and reaches that point, it is interrupted and the debugging routine is activated. This gives the user a chance to examine memory and register contents. When the user is ready to continue executing the

program being debugged, the debugging routine restores the saved instruction that was a location i+1 and executes a Return-from-interrupt instruction.

Privilege Exception:

To protect the operating system of a computer from being corrupted by user programs, certain instructions can be executed only while the processor is in supervisor mode. These are called privileged instructions. For example, when the processor is running in the user mode, it will not execute an instruction that changes the priority level of the processor or that enables a user program to access areas in the computer memory that have been allocated to other users. An attempt to execute such an instruction will produce a privilege exceptions, causing the processor to switch to the supervisor mode and begin executing an appropriate routine in the operating system.