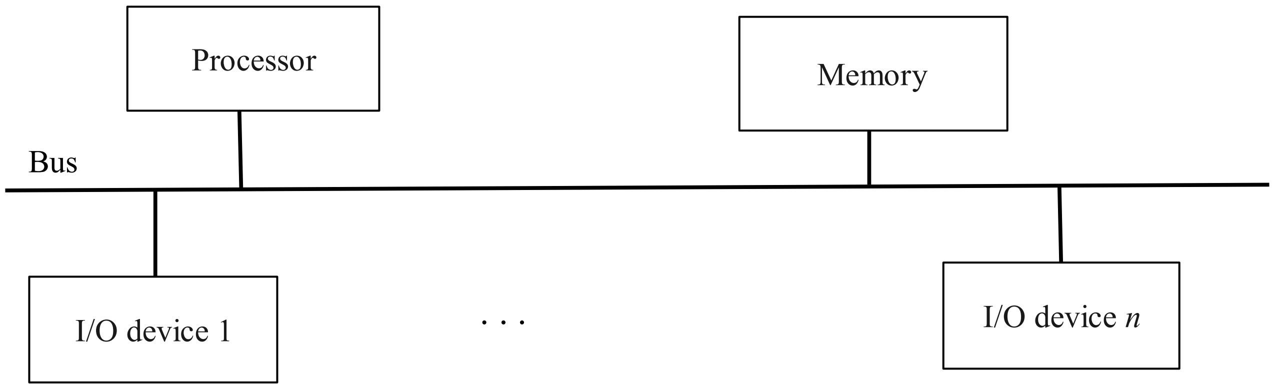
Module 2

Input output organization

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. 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 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.



A single-bus Structure

Memory-mapped I/O:

→ memory and I/O share the same address space.
→ reading and writing are similar to memory read/write. (same instructions are used)
→ uses same memory read and write signals.
→ Most computer systems use memory-mapped I/O.

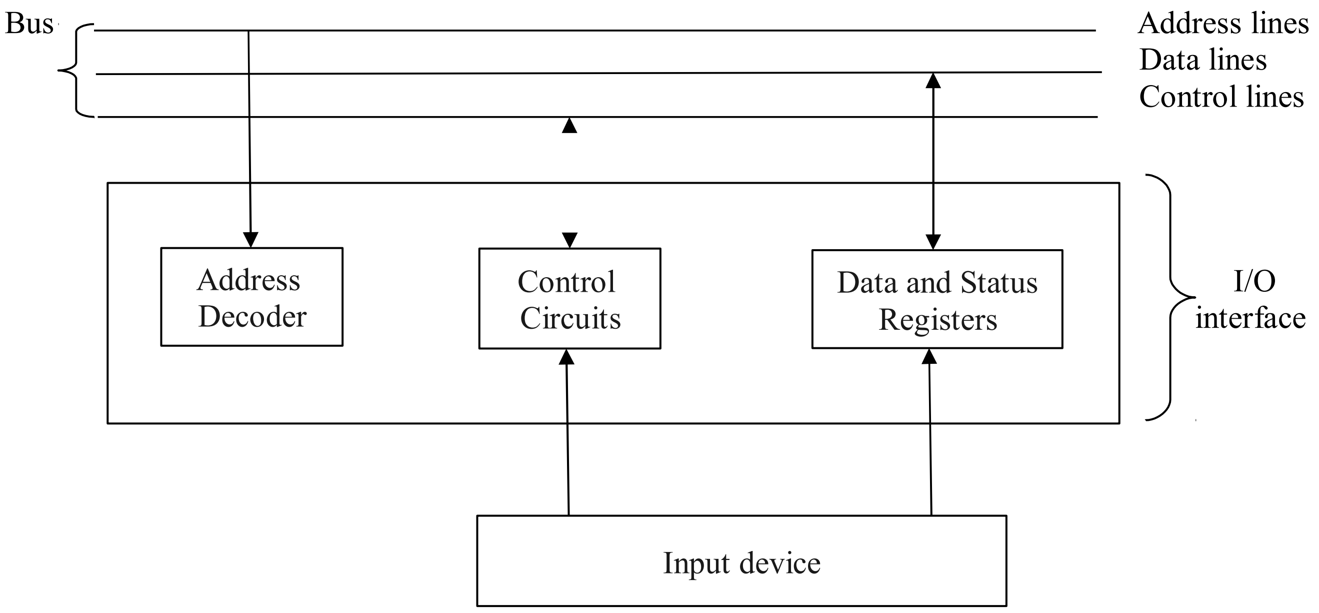
I/O mapped I/O:

→ uses separate address space.
→ less address lines are used for accessing I/O.
→ separate I/O read and write signals are needed.
→ In and Out instructions are used to perform I/O transfers.
→ Pentium supports I/O mapped.

In **program-controlled I/O**, the processor repeatedly checks a status flag to achieve the required synchronization between the processor and an input or output device. The processor polls the device.

I/O Interface

- 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.



Registers in keyboard and display interfaces

DATAIN								
DATAOUT								
STATUS					DIRQ	KIRQ	SOUT	SIN
CONTROL					DEN	KEN		
	7	6	5	4	3	2	1	0

Three types of data transfers:

1. Program-controlled
2. Interrupts
3. DMA (Direct Memory Access)

A program that reads one line from keyboard, stores it in memory buffer, and echoes it back to the display. Below program is an example for **program controlled I/O**.

```
         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 character and advance the 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 input line.
```

Interrupts:

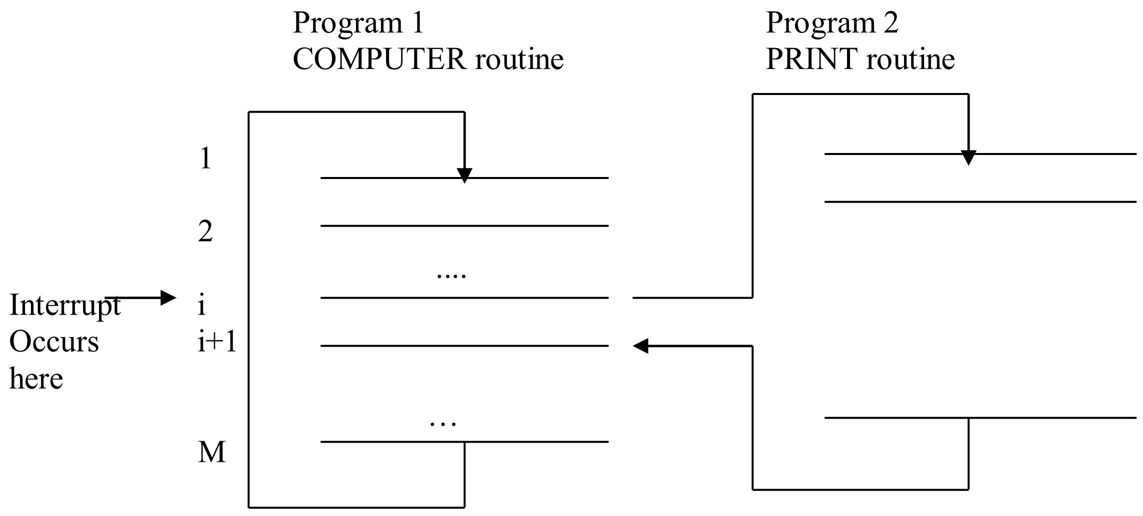
INT: Interrupt is a hardware signal sent by I/O device when it is become ready to alert the processor.

INTR: At least one of the control lines called interrupt request line is usually dedicated for this purpose.

INTA: interrupt-acknowledge signal is a special signal issued by the processor to the device that its request has been recognized so that it may remove its interrupt request signal.

ISR: The routine executed in response to an interrupt request is called the interrupt-service routine.

Assume that an interrupt request arrives during execution of instruction i



Transfer of control using 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. 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.

Saving and restoring registers during interrupt

- 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.
- 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.
- 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.**
- All registers are saved automatically by the processor hardware at the time an interrupt request is accepted. The data saved are restored to their respective registers as part of the execution of the Return-from interrupt instruction.

INTERRUPT HARDWARE: -

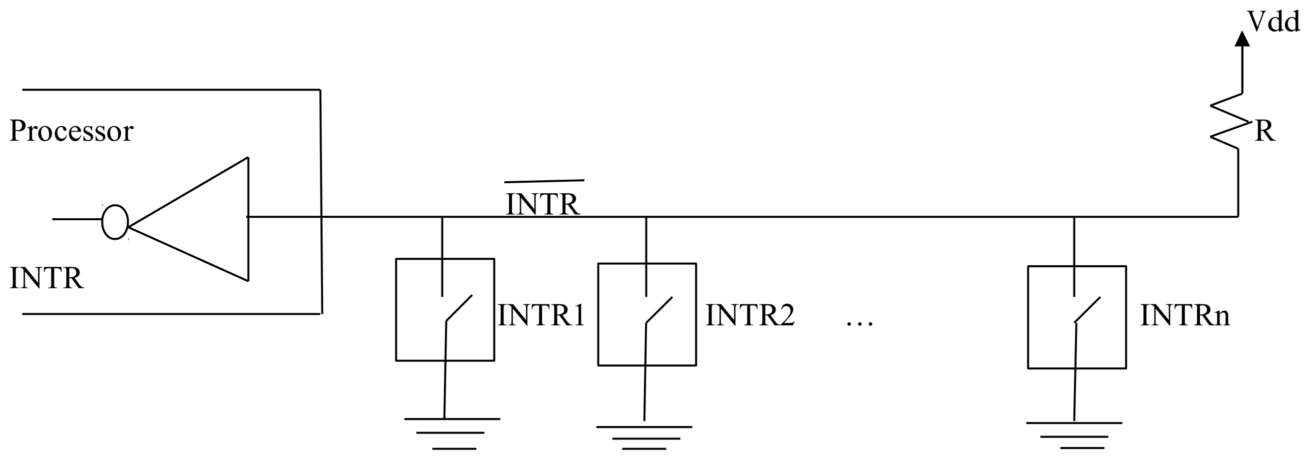
- A single interrupt-request line may be used to serve n devices as depicted.
- All devices are connected to the line via switches to ground.
- To request an interrupt, a device closes its associated switch.
- Thus, if all interrupt-request signals $INTR_1$ to $INTR_n$ are inactive, that is, if all switches 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,

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

It is customary to use the complemented form, $I\overline{NTR}$, to name the interrupt-request signal on the common line, because this signal is active when in the low-voltage state.

Open-collector are used to drive the $\overline{INTR}$ line. The output of the open-collector gate is equivalent to the switch to ground that is open when the gate's input is in the 0 state and closed when it is in the state 1.

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.



ENABLING AND DISABLING INTERRUPTS:

When a device activates the interrupt-request signal, it keeps this signal activated until it learns that the processor has accepted its request. This may lead to recursive requests from the same device. This can be handled using 3 techniques, namely:

- Ignore
- Disable
- Use special hardware line

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.

Th**e 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 (PSW- Processor Status Word)** , 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.

HANDLING MULTIPLE DEVICES: -

When a request is received over the common interrupt-request line, additional information is needed to identify the 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 IRQ bit to 1, which is in its status register. 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.

Vectored Interrupts: -

- Used 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 refer 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.
- The processor reads this address called the interrupt vector and loads it into the PC

When the device sends an interrupt request the processor may not be ready to receive the vector code immediately. It must first complete the execution of the current instruction, which may require the use of the bus. When the processor is ready to receive the vector code, it activates the INTA line. The I/O device responds by sending it interrupt vector code and turning of the INTR signal.

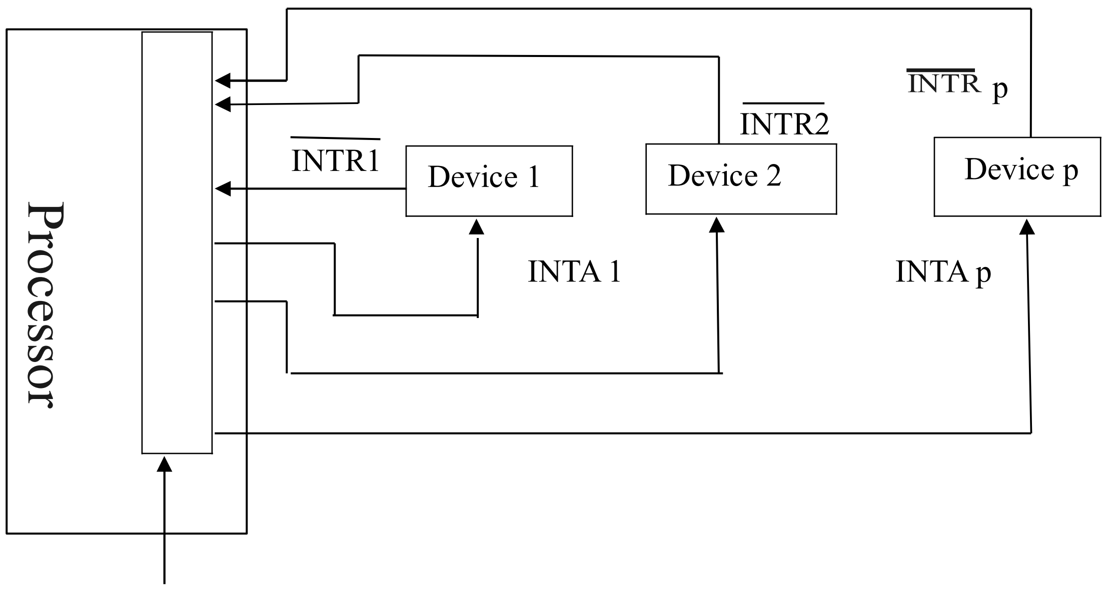
Interrupt Nesting: -

- Interrupts should be disabled during the execution of an ISR, to ensure that a request from one device will not cause more than one interruption.
- Execution of a given ISR, once started, always continues to completion before the processor accepts an interrupt request from a second device.
- But an interrupt request from a high-priority device should be accepted while the processor is servicing another request from a lower-priority device.

To implement this scheme, 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.

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.



Processor
$\overline{\text{INTR1}}$
$\overline{\text{INTR2}}$
$\overline{\text{INTR}}$ p
Device 1
Device 2
Device p
INTA 1
INTA p
Priority arbitration Circuit

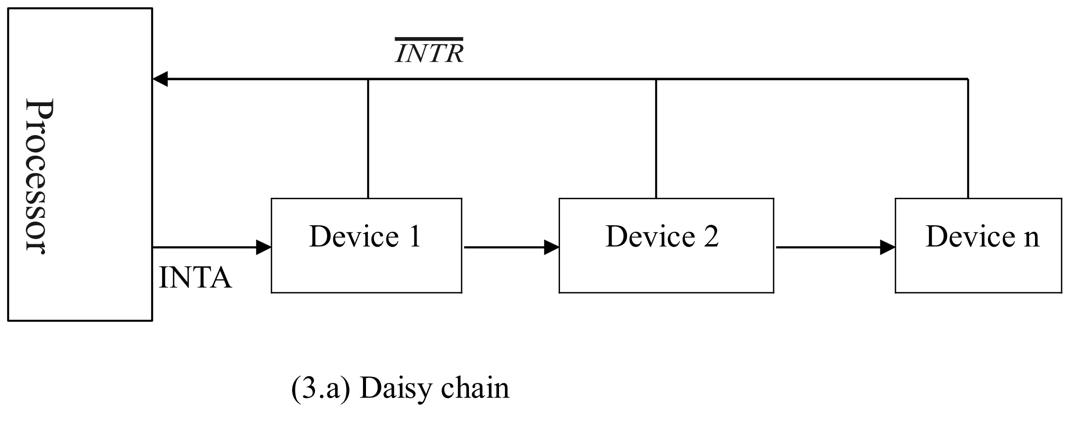
fig 2

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

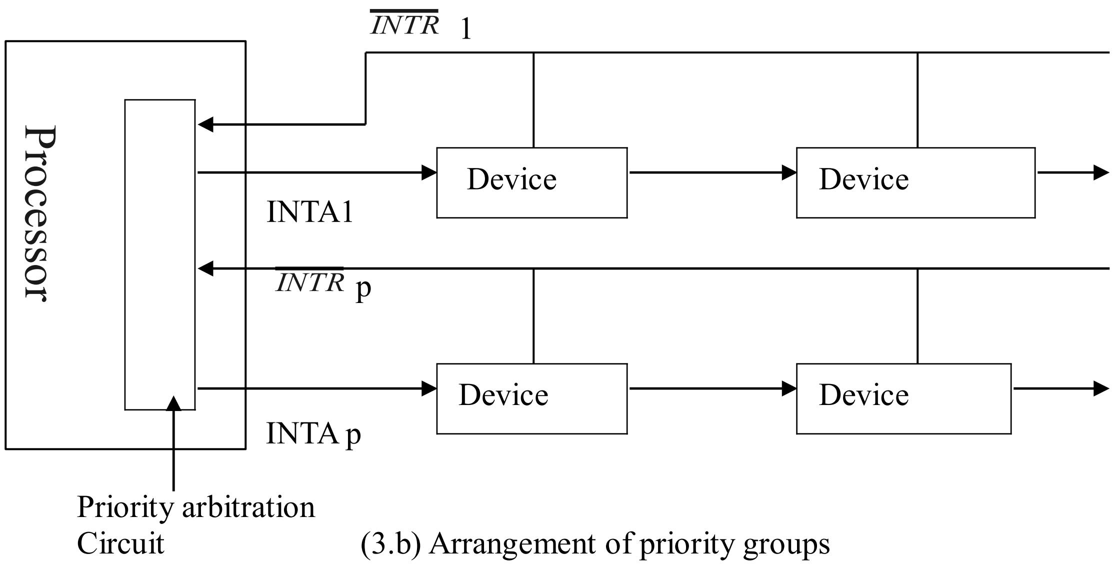
Simultaneous Requests: -

Consider the problem of simultaneous arrivals of interrupt requests from two or more devices. 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 $\overline{\text{INTR}}$ 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.



(3.a) Daisy chain



(3.b) Arrangement of priority groups

When several devices raise an interrupt request and the $\overline{INTR}$ 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. 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.

Exceptions:-

An interrupt is an event that causes the execution of one program to be suspended and the execution of another program to begin. However, the interrupt mechanism is used in several 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.

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.

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 the same actions as when a hardware interrupt request is received.

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.