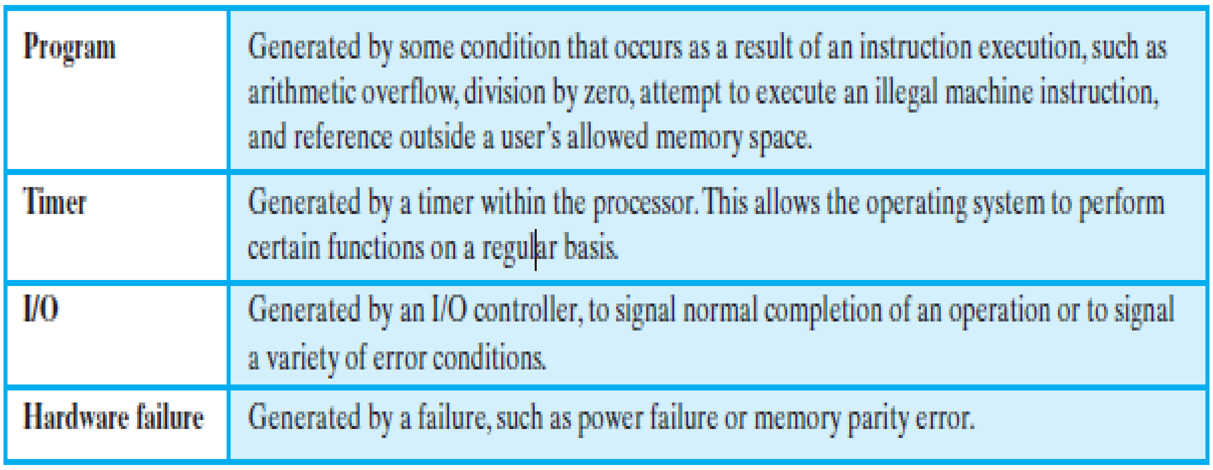
INTERRUPTS:

- An interrupt is defined as hardware or software generated event external to the currently executing process that affects the normal flow of the instruction execution.
- Interrupts are provided primarily as a way to improve processor utilization

Classes of Interrupts:



Program	Generated by some condition that occurs as a result of an instruction execution, such as arithmetic overflow, division by zero, attempt to execute an illegal machine instruction, and reference outside a user's allowed memory space.
Timer	Generated by a timer within the processor. This allows the operating system to perform certain functions on a regular basis.
I/O	Generated by an I/O controller, to signal normal completion of an operation or to signal a variety of error conditions.
Hardware failure	Generated by a failure, such as power failure or memory parity error.

Example: Consider a processor that executes a user application. In figure (a) the user program performs a series of WRITE calls interleaved with processing.

- The WRITE calls are the call to an I/O routine that is a system utility and that will perform the actual I/O operation. The I/O program consists of three sections:

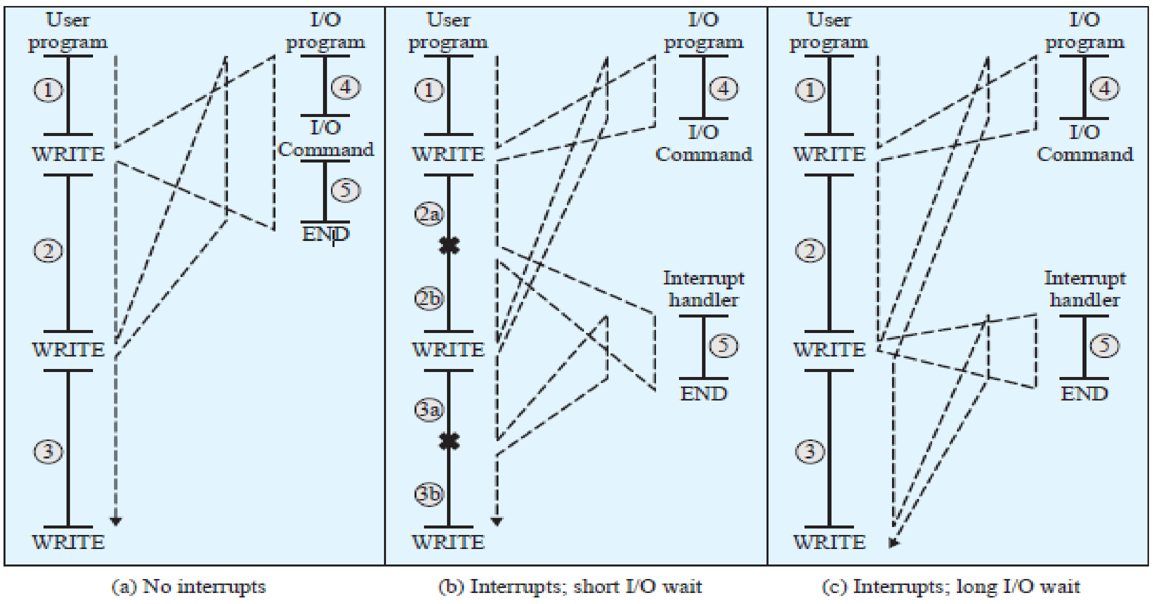
i) A sequence of instructions (4) to prepare for the actual I/O operation. This may include copying the data to be output into a special buffer and preparing the parameters for a device command.

ii) The actual I/O command. Without the use of interrupts, once this command is issued, the program must wait for the I/O device to perform the requested function. The program might wait by simply repeatedly performing a test operation to determine if the I/O operation is done.

iii) A sequence of instructions (5) to complete the operation. This may include setting a flag indicating the success or failure of the operation.

- After the first WRITE instruction is encountered, the user program is interrupted and execution continues with the I/O program.
- After the I/O program execution is complete, execution resumes in the user program immediately following the WRITE instruction.

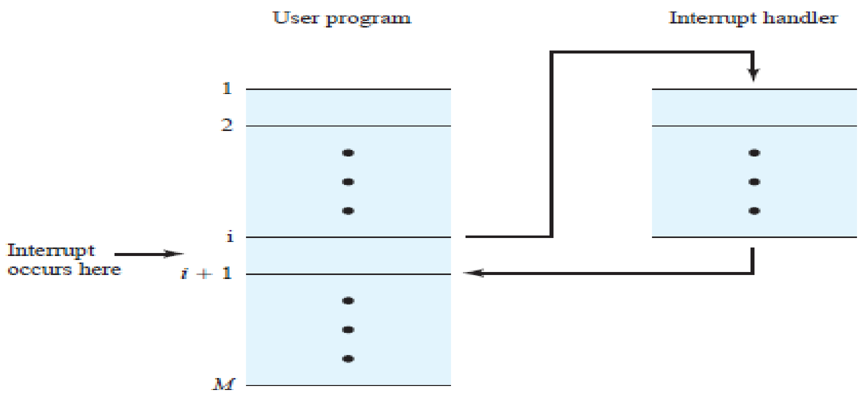
.



Program Flow of Control without and with Interrupts

Interrupts and the Instruction Cycle:

- ❖ With interrupts, the processor can be engaged in executing other instructions while an I/O operation is in progress.
- ❖ When the processor encounters the WRITE instruction the I/O program is invoked that consists only of the preparation code and the actual I/O command. After these few instructions have been executed, control returns to the user program.
- ❖ Meanwhile, the external device is busy accepting data from computer memory and printing it. This I/O operation is conducted concurrently with the execution of instructions in the user Program.
- ❖ When the external device becomes ready to accept more data from the processor, the I/O module for that external device sends an interrupt request signal to the processor.
- ❖ The processor responds by suspending operation of the current program. This process of branching off to a routine to service that particular I/O device is known as an **interrupt handler** and resuming the original execution after the device is serviced.



Transfer of control via Interrupts

- ❖ To accommodate interrupts, an interrupt stage is added to the instruction cycle. In the interrupt stage, the processor checks to see if any interrupts have occurred, indicated by the presence of an interrupt signal.
- ❖ If no interrupts are pending, the processor proceeds to the fetch stage and fetches the next instruction of the current program.
- ❖ If an interrupt is pending, the processor suspends execution of the current program and executes an interrupt-handler routine. This routine determines the nature of the interrupt and performs whatever actions are needed.

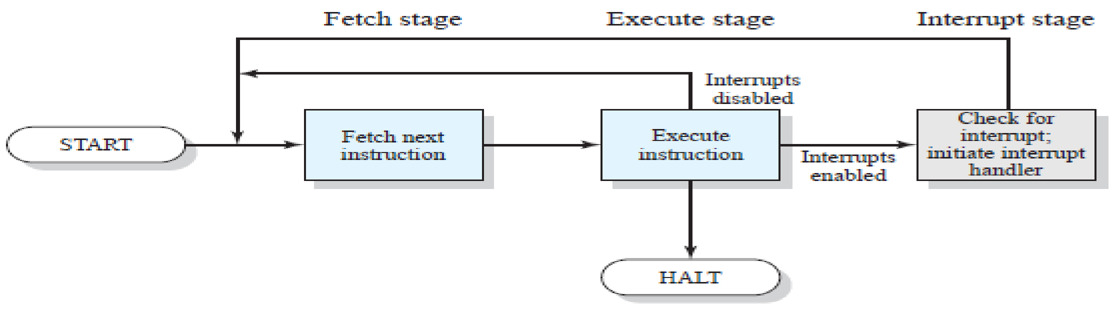


Figure 1.7 **Instruction Cycle with Interrupts**

Interrupt Processing:

- ❖ An interrupt triggers a number of events, both in the processor hardware and in software. When an I/O device completes an I/O operation, the following hardware events occurs:

1. The device issues an interrupt signal to the processor.
2. The processor finishes execution of the current instruction before responding to the interrupt.
3. The processor tests for a pending interrupt request, determines that there is one, and sends an acknowledgment signal to the device that issued the interrupt. The acknowledgment allows the device to remove its interrupt signal.
4. The processor next needs to prepare to transfer control to the interrupt routine. It saves the program status word (PSW) and the location of the next instruction to be executed, which is contained in the program counter. These can be pushed onto a control stack.
5. The processor then loads the program counter with the entry location of the interrupt-handling routine that will respond to this interrupt.

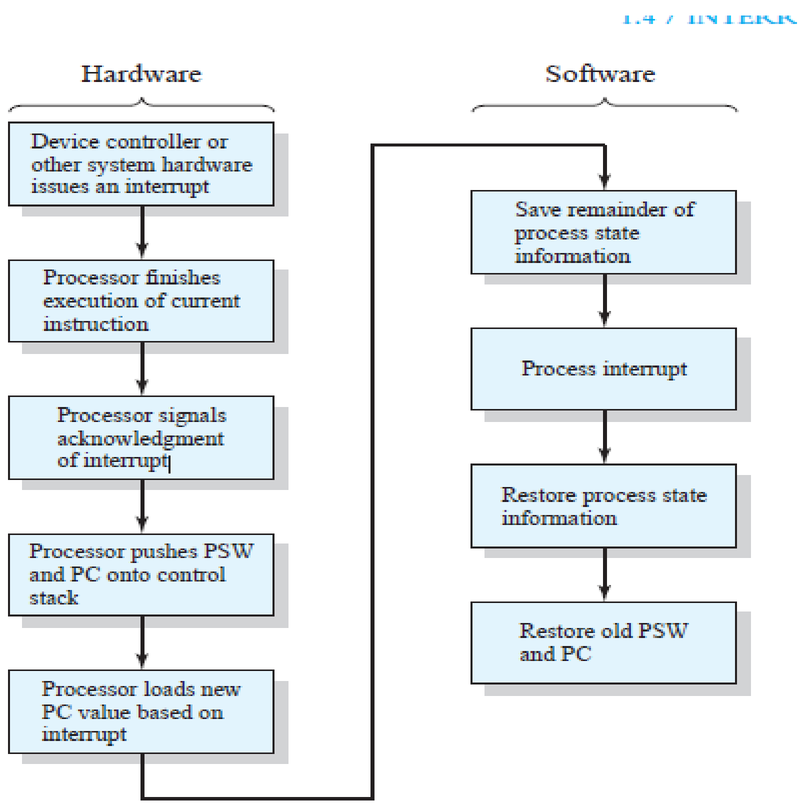
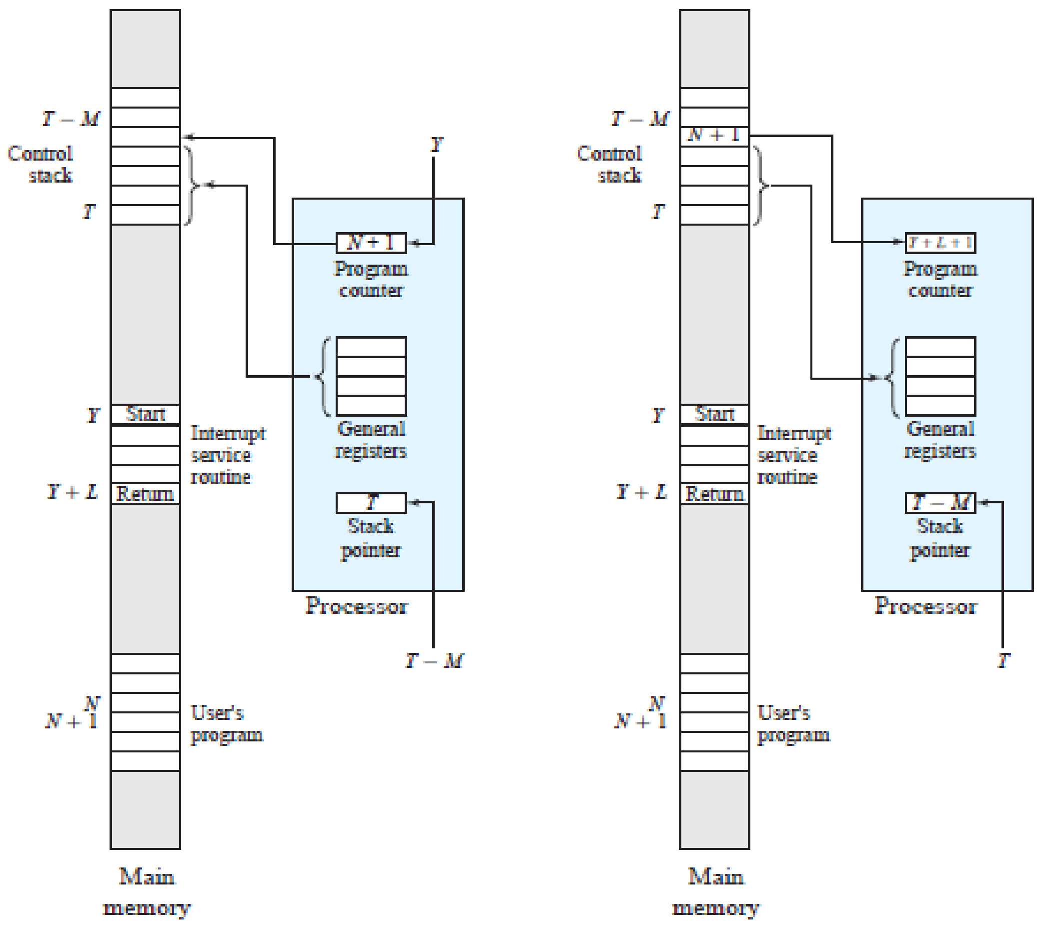

Hardware
Software
Device controller or other system hardware issues an interrupt
Processor finishes execution of current instruction
Processor signals acknowledgment of interrupt
Processor pushes PSW and PC onto control stack
Processor loads new PC value based on interrupt
Save remainder of process state information
Process interrupt
Restore process state information
Restore old PSW and PC

Figure 1.10 **Simple Interrupt Processing**

6. Once the program counter has been loaded, the processor proceeds to the next instruction cycle, which begins with an instruction fetch. The contents of the processor registers need to be saved, because these registers may be used by the interrupt handler. So all of these values, plus any other state information, need to be saved.
7. The interrupt handler may now proceed to process the interrupt.
8. When interrupt processing is complete, the saved register values are retrieved from the stack and restored to the registers
9. The final act is to restore the PSW and program counter values from the stack. As a result, the next instruction to be executed will be from the previously interrupted program.

- The following is an example for a user program that is interrupted after the instruction at location N.
- The contents of all of the registers plus the address of the next instruction (N + 1), a total of M words, are pushed onto the control stack.
- The stack pointer is updated to point to the new top of stack, and the program counter is updated to point to the beginning of the interrupt service routine.



(a) Interrupt occurs after instruction at location N

(b) Return from interrupt

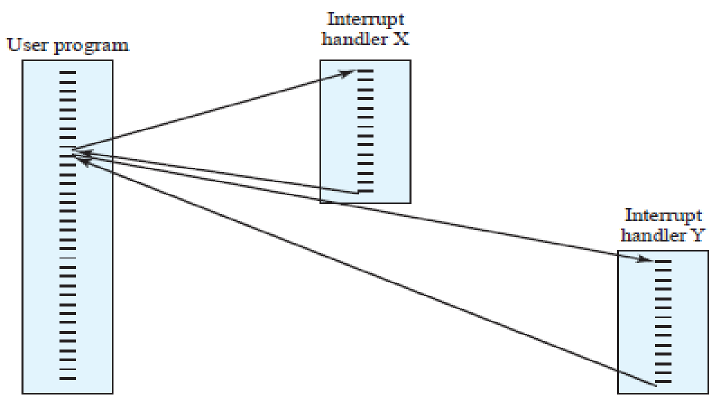
Multiple Interrupts:

- ❖ One or more interrupts can occur while an interrupt is being processed. This is called as Multiple Interrupts.

- ❖ Two approaches can be taken to dealing with multiple interrupts.
  - i) Sequential interrupt processing
  - ii) Nested interrupt processing

Sequential interrupt processing:

- ❖ The first approach is to disable interrupts while an interrupt is being processed. A disabled interrupt simply means that the processor ignores any new interrupt request signal.

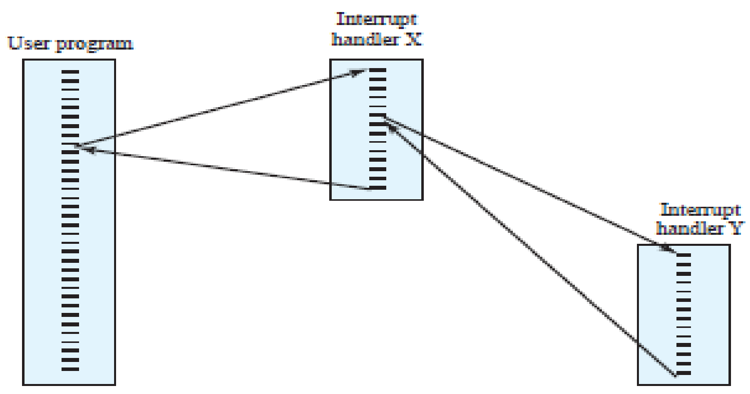


(a)Sequential Interrupt processing.

- ❖ If an interrupt occurs during this time, it generally remains pending and will be checked by the processor after the processor has reenabled interrupts.
- ❖ Thus, when a user program is executing and an interrupt occurs, interrupts are disabled immediately. After the interrupt-handler routine completes, interrupts are reenabled before resuming the user program and the processor checks to see if additional interrupts have occurred.
- ❖ This approach is simple as interrupts are handled in strict sequential order.
- ❖ The drawback to this approach is that it does not take into account relative priority or time-critical needs.

Nested interrupt processing:

- ❖ A second approach is to define priorities for interrupts and to allow an interrupt of higher priority to cause a lower-priority interrupt handler to be interrupted.



(b) Nested interrupt processing

Figure 1.12 **Transfer of Control with Multiple Interrupts**

- ❖ Let us consider a system with three I/O devices. A printer, a disk, and a communications line, with increasing priorities of 2, 4, and 5, respectively. A user program begins at t=0. At t=10, a printer interrupt occurs.
- ❖ While this routine is still executing, at t=15, a communications interrupts occur. Because the communications line has highest priority than the printer, the interrupt request is honored.
- ❖ The printer ISR is interrupted, its state is pushed onto the stack and the execution continues at the communications ISR. While this routine is executing an interrupt

occurs at t=20.This interrupt is of lower priority it is simply held and the communications ISR runs to the completion.

- When the communications ISR is complete at t=25, the previous processor state is restored which the execution of the printer ISR. However, before even a single instruction in that routine can be executed the processor honors the higher priority disk interrupt and transfers control to the disk ISR. Only when that routine completes (t= 35) the printer ISR is resumed. When the Printer ISR completes at t=40then finally the control returns to the user program.

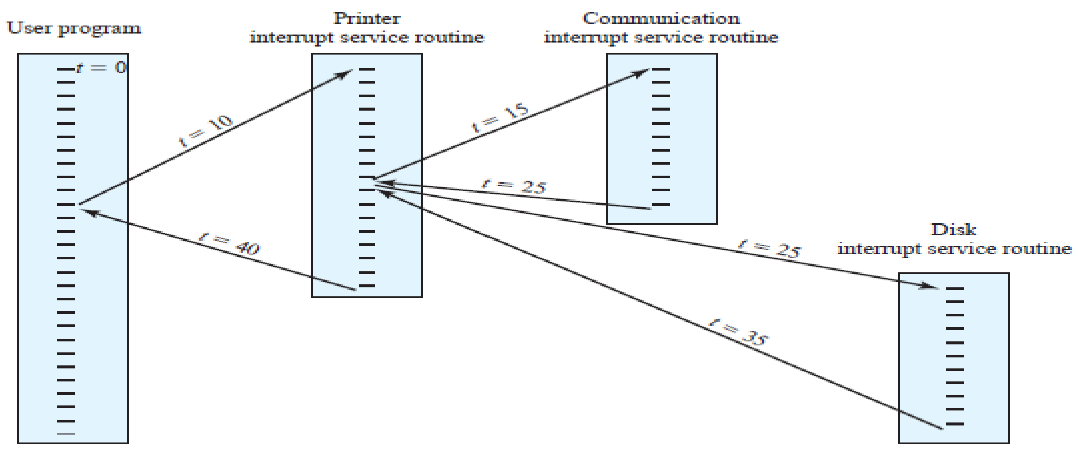

User program
Printer interrupt service routine
Communication interrupt service routine
Disk interrupt service routine
t = 0
t = 10
t = 15
t = 25
t = 25
t = 35
t = 40

Figure 1.13 Example Time Sequence of Multiple Interrupts

Multiprogramming:

- With the use of interrupts, a processor may not be used very efficiently. If the time required to complete an I/O operation is much greater than the user code between I/O calls then the processor will be idle much of the time.
- A solution to this problem is to allow multiple user programs to be active at the same time. This approach is called as multiprogramming.
- When a program has been interrupted, the control transfers to an interrupt handler, once the interrupt- handler routine has completed, control may not necessarily immediately be returned to the user program that was in execution at the time.
- Instead, control may pass to some other pending program with a higher priority. This concept of multiple programs taking turns in execution is known as multiprogramming.