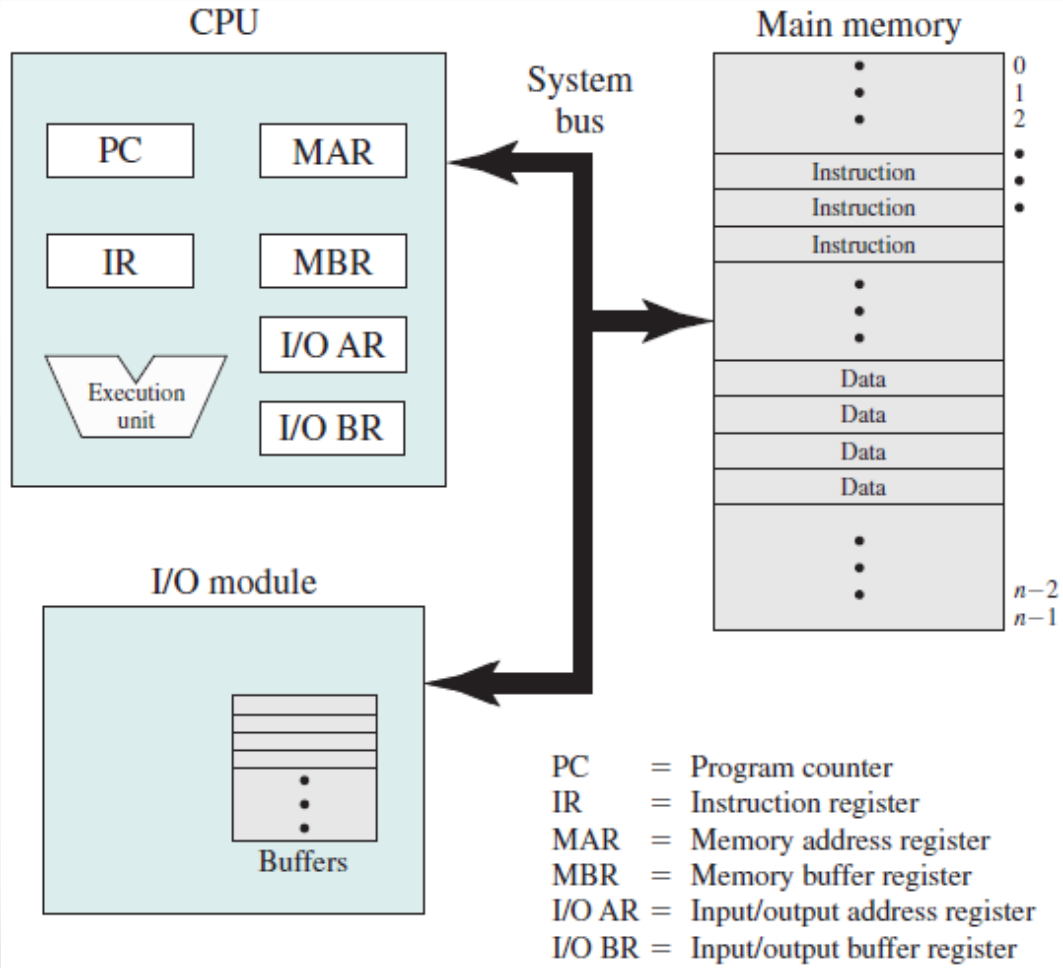
COMPUTER SYSTEM OVERVIEW

NEED TO UNDERSTAND COMPUTER SYSTEM COMPONENTS

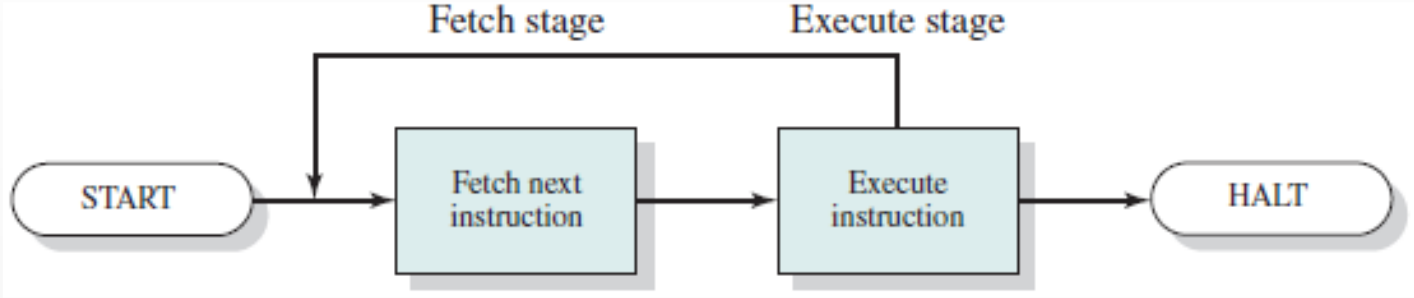
- As an Operating System manages hardware resources in order to provide service to computer system users. It is therefore important to have some understanding of the underlying computer system hardware before we begin our examination of operating systems.

BASIC ELEMENTS



INSTRUCTION EXECUTION

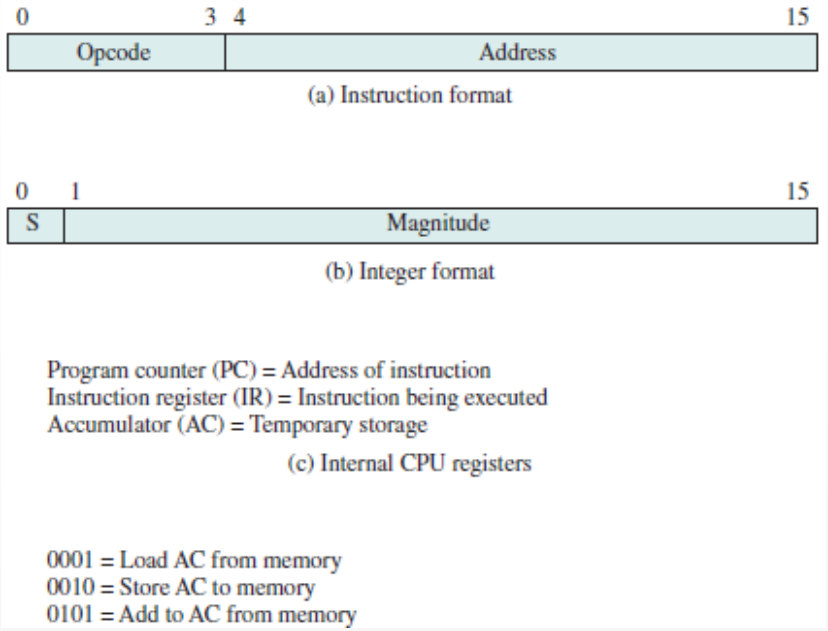
- In its simplest form, instruction processing (instruction cycle) consists of two steps:
  1. The processor reads (***fetches***) instructions from memory one at a time
  2. ***executes*** each instruction



- Program execution consists of repeating the process of instruction fetch and instruction execution.
- the program counter (PC) holds the address of the next instruction to be fetched.
- Unless instructed otherwise, the processor always increments the PC after each instruction fetch so that it will fetch the next instruction in sequence.

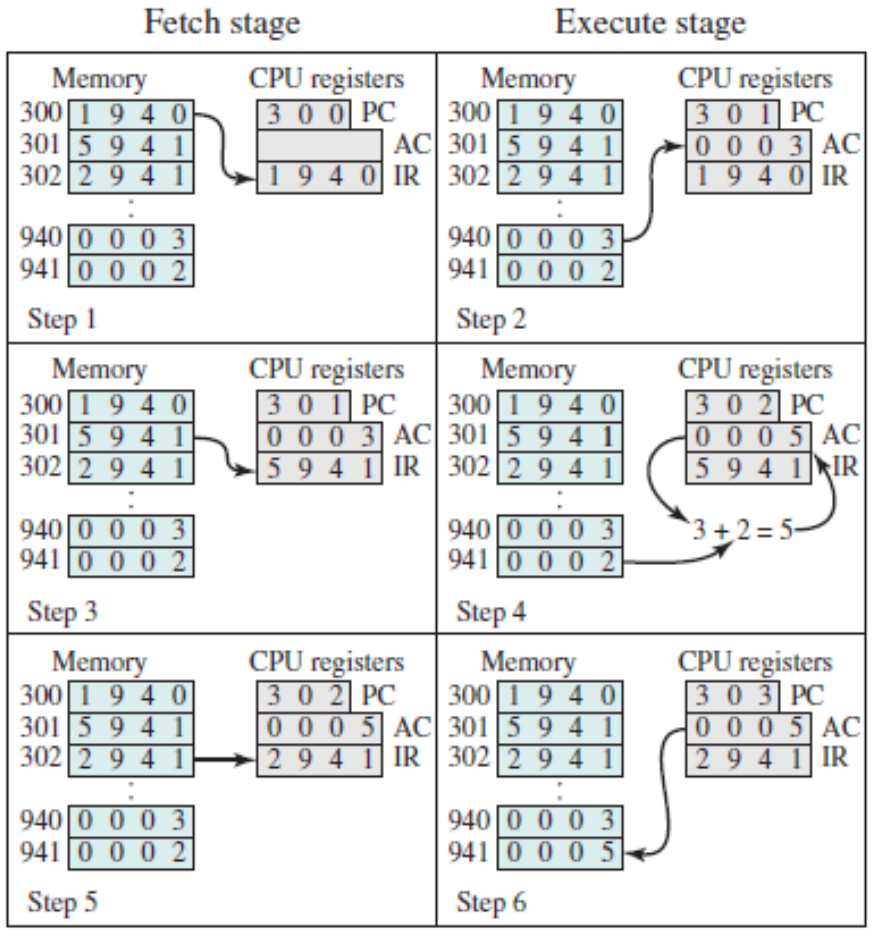
INSTRUCTION EXECUTION

- The fetched instruction is loaded into the **instruction register (IR).**
- The instruction contains bits that specify the action the processor is to take. The processor interprets the instruction and performs the required action. In general, these actions fall into four categories:
- **Processor-memory**
- **Processor-I/O**
- **Data processing:**
- **Control:** An instruction may specify that the sequence of execution be altered.
    - For example, the processor may fetch an instruction from location 149, which specifies that the next instruction be from location 182. The processor sets the program counter to 182.



INSTRUCTION EXECUTION

- Figure shows the partial program execution, showing the relevant portions of memory and processor registers.
- The program fragment shown adds the contents of the memory word at address 940 to the contents of the memory word at address 941 and stores the result in the latter location.
- Three instructions, which can be described as three fetch and three execute stages, are required.



INTERRUPTS

- Virtually all computers provide a mechanism by which other modules (I/O, memory) may interrupt the normal sequencing of the processor.

- Most I/O devices are much slower than the processor. Interrupts are provided primarily as a way to improve processor utilization.

- **Example 1** Suppose that the processor is transferring data to a printer using the instruction cycle scheme. After each write operation, the processor must pause and remain idle until the printer catches up. The length of this pause may be on the order of many thousands or even millions of instruction cycles. Clearly, this is a very wasteful use of the processor.

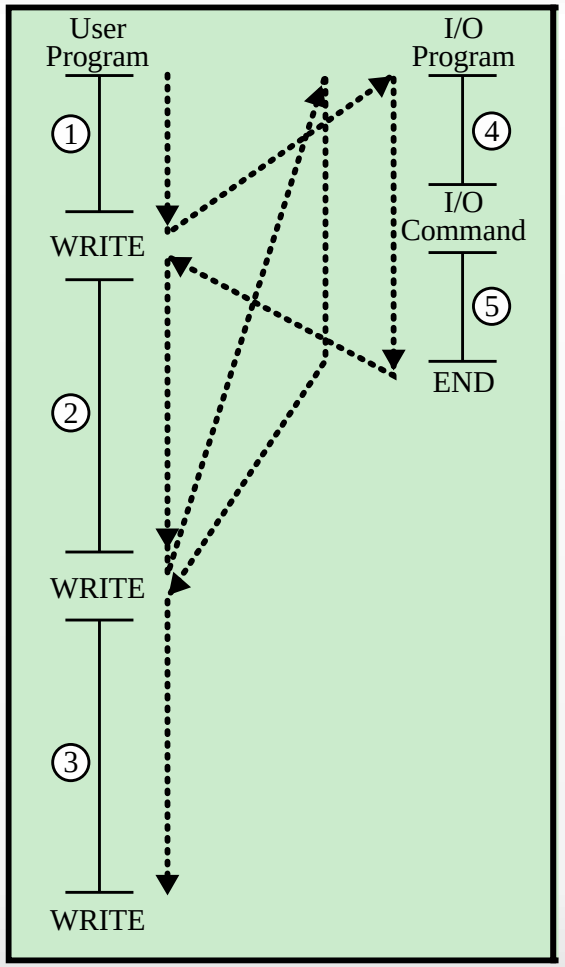
- **Example 2:** consider a PC that operates at 1 GHz, which would allow roughly 109 instructions per second.2 A typical hard disk has a rotational speed of 7200 revolutions per minute for a half-track rotation time of 4 ms, which is 4 million times slower than the processor.

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.

INTERRUPTS EXAMPLE

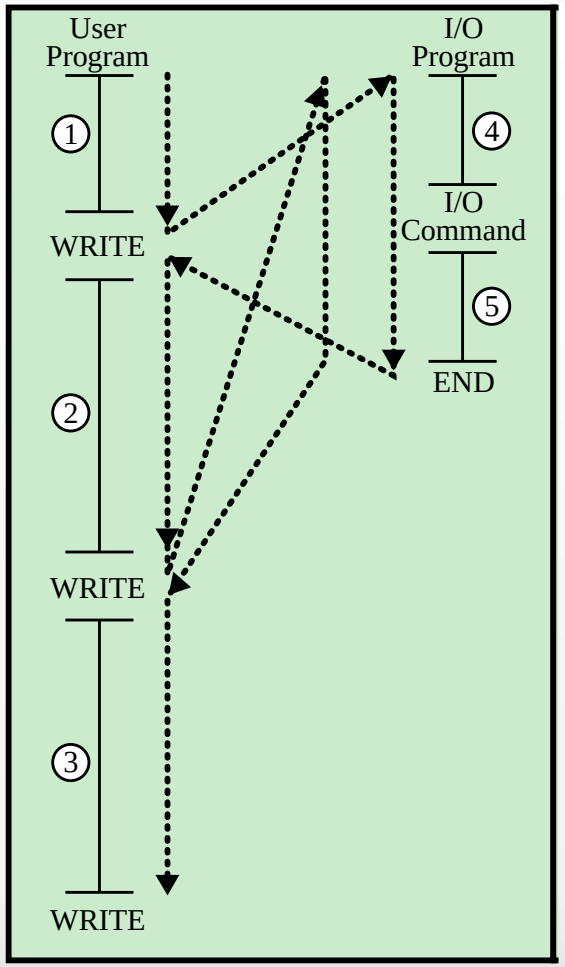
- The user program performs a series of WRITE calls interleaved with processing. The solid vertical lines represent segments of code in a program. Code segments 1, 2, and 3 refer to sequences of instructions that do not involve I/O. The WRITE calls are to an I/O routine that is a system utility and that will perform the actual I/O operation. The I/O program consists of three sections:
  - A sequence of instructions, labeled 4 in the figure, 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.
  - 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 (or periodically check the status, or poll, the I/O device). The program might wait by simply repeatedly performing a test operation to determine if the I/O operation is done.
  - A sequence of instructions, labeled 5 in the figure, to complete the operation. This may include setting a flag indicating the success or failure of the operation.



(a) No interrupts

INTERRUPTS EXAMPLE

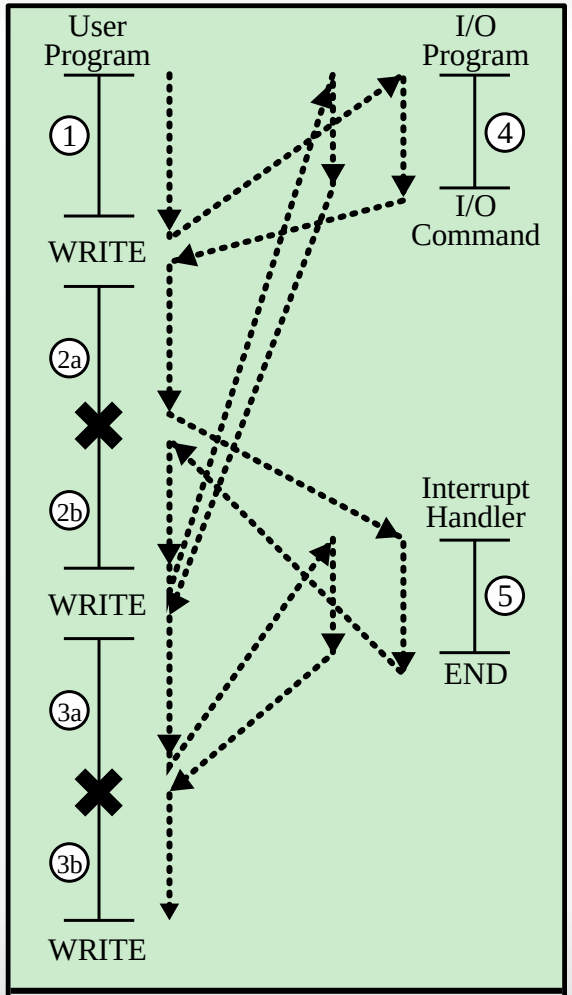
- The dashed line represents the path of execution followed by the processor; that is, this line shows the sequence in which instructions are executed. Thus, after the first WRITE instruction is encountered, the user program is interrupted and execution continues with the I/O program.
- After the I/O program execution is complete, execution resumes in the user program immediately following the WRITE instruction.
- Because the I/O operation may take a relatively long time to complete, the I/O program is hung up waiting for the operation to complete; hence, the user program is stopped at the point of the WRITE call for some considerable period of time.



(a) No interrupts

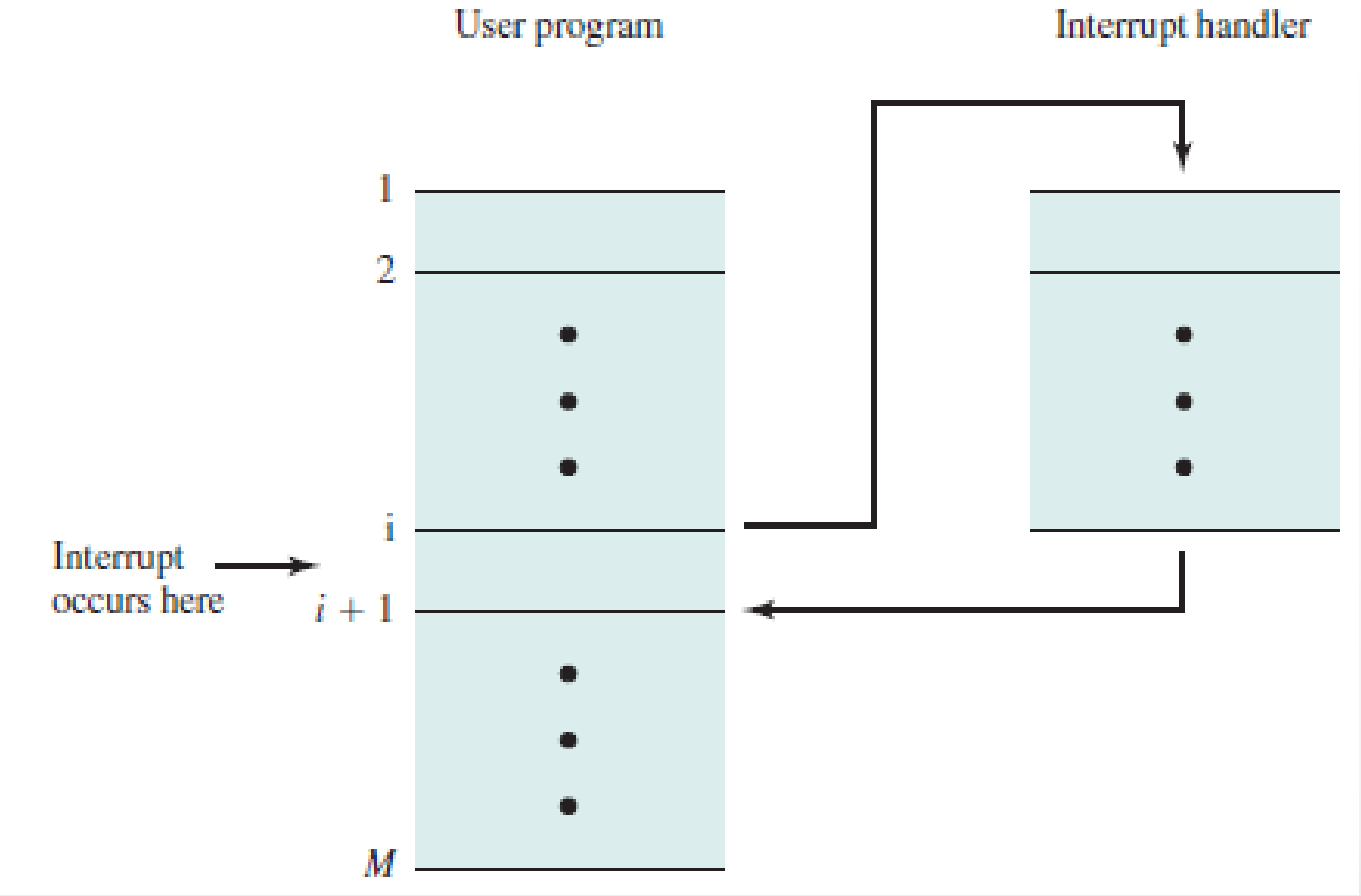
INTERRUPTS EXAMPLE

- **With interrupts**, the processor can be engaged in executing other instructions while an I/O operation is in progress.
- As before, the user program reaches a point at which it makes a **system call** (WRITE call). The I/O program that is invoked in this case 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 be serviced, that is, when it is 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; branching off to a routine to service that particular I/O device, known as an interrupt handler; and resuming the original execution after the device is serviced.
- Note that an interrupt can occur at any point in the main program, not just at one specific instruction.



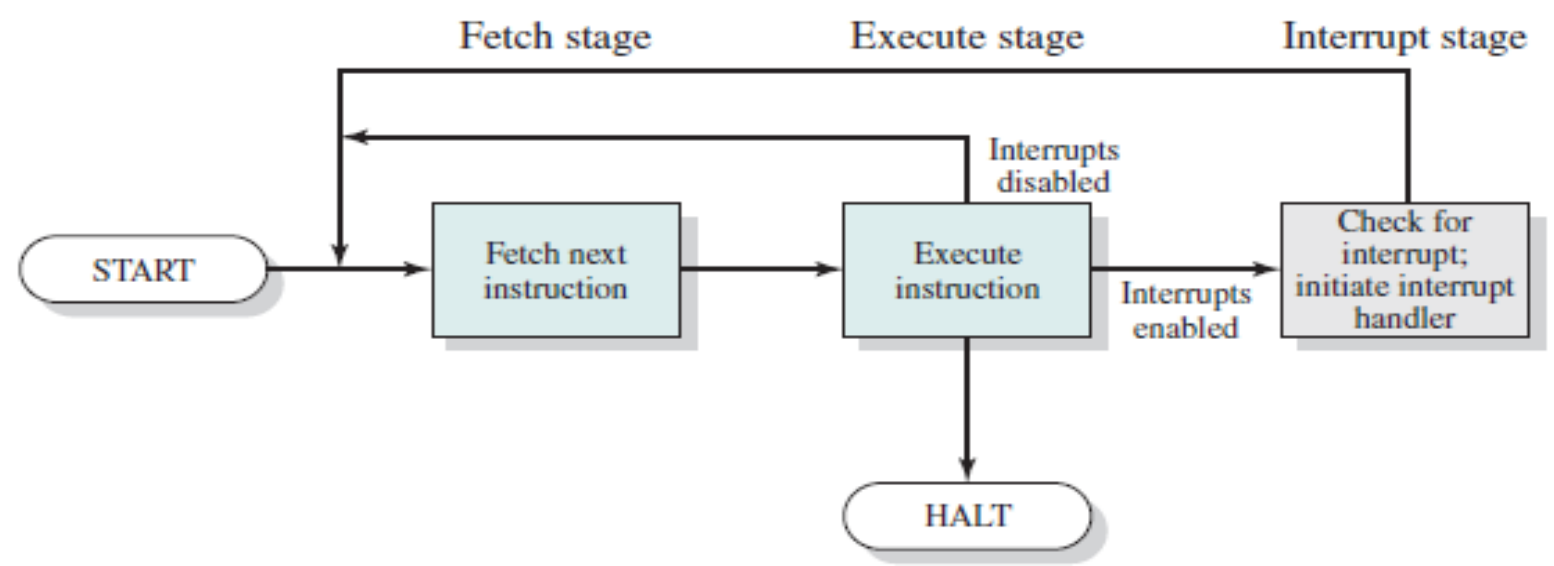
(b) Interrupts; short I/O wait

TRANSFER OF CONTROL VIA INTERRUPTS

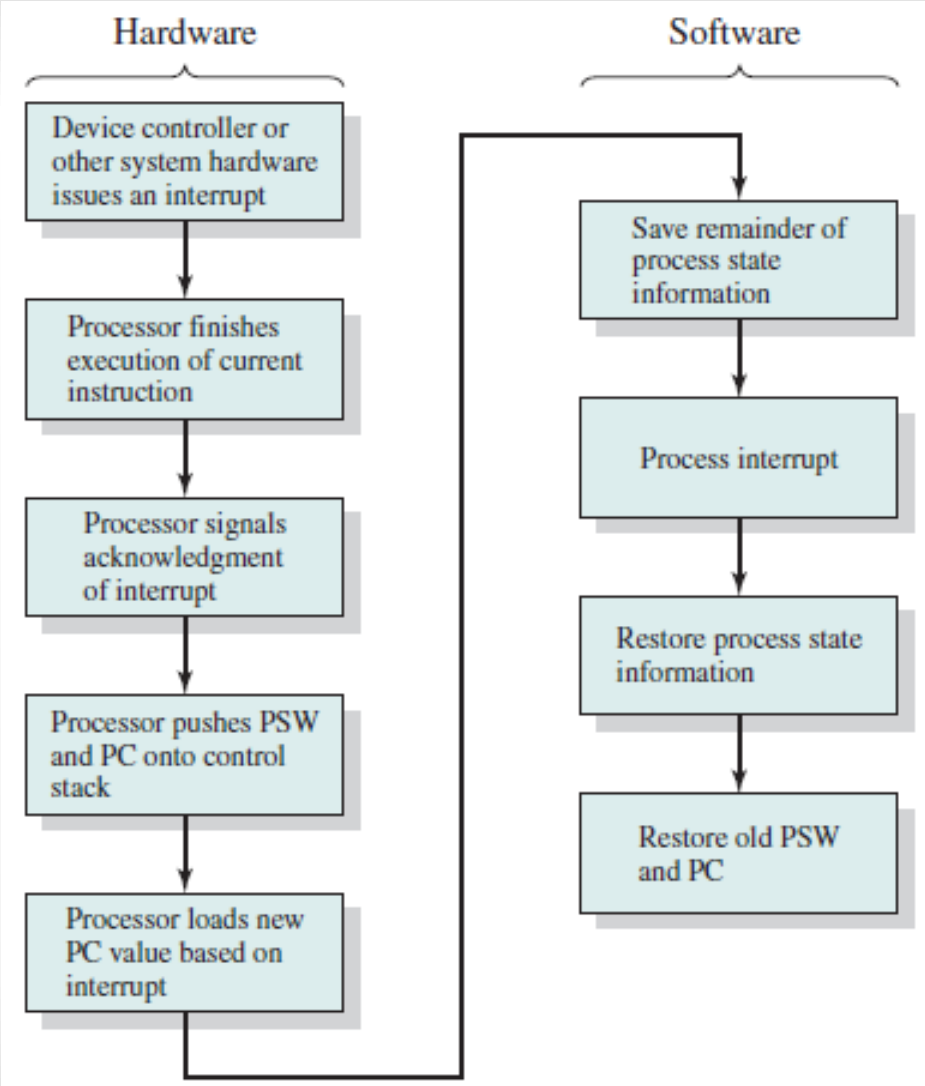


INSTRUCTION CYCLE WITH INTERRUPTS

- 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**.
- The interrupt-handler routine is generally part of the OS, that determines the nature of the interrupt and performs whatever actions are needed.



INTERRUPT PROCESSING



MULTIPLE INTERRUPTS

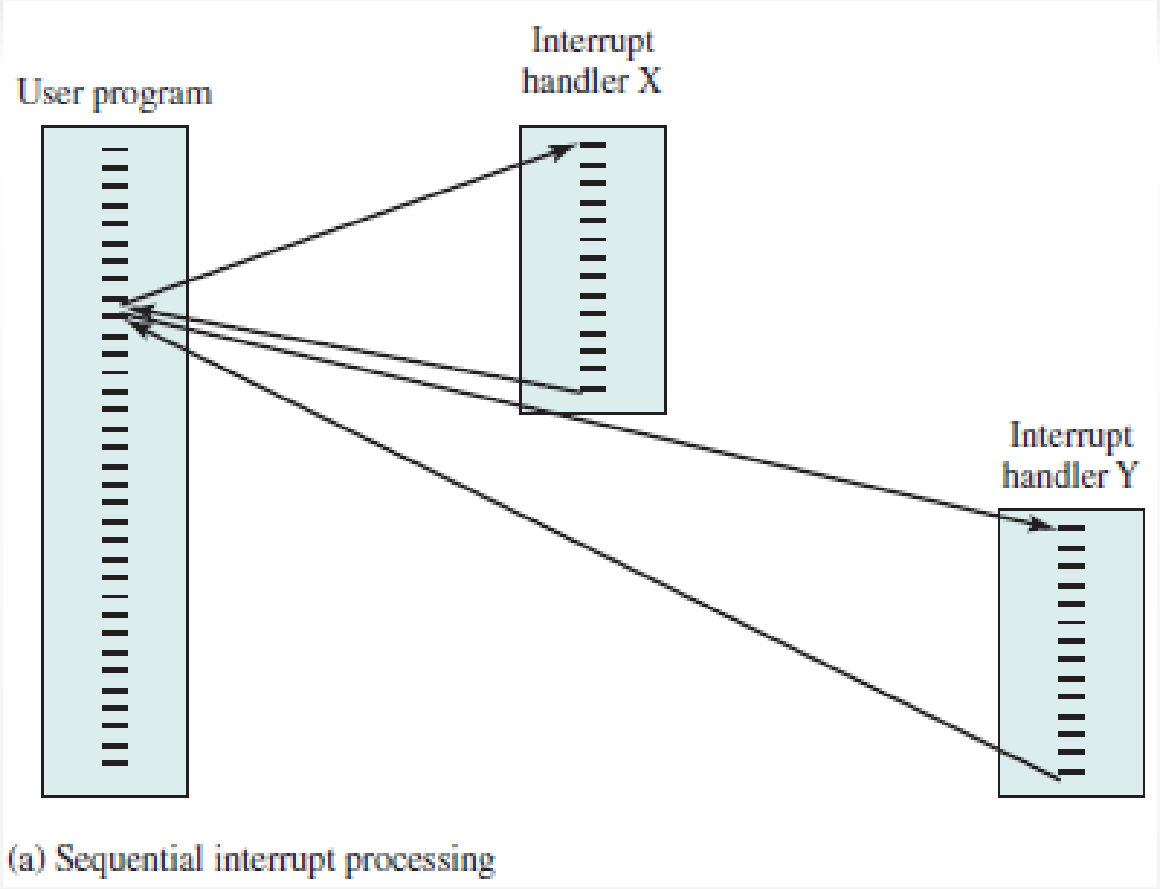
An interrupt occurs while another interrupt is being processed

- e.g. receiving data from a communications line and printing results at the same time

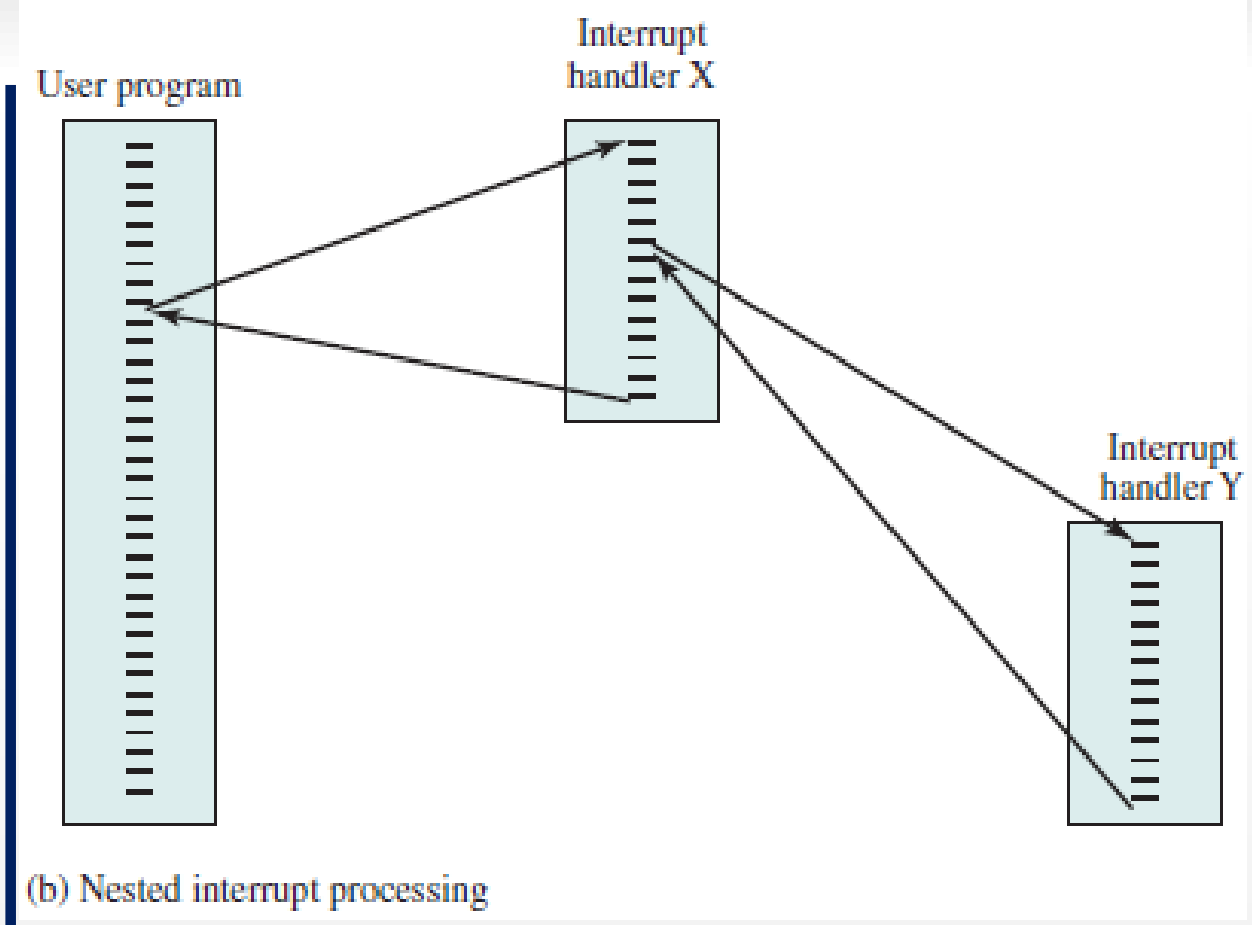
Two approaches:

- disable interrupts while an interrupt is being processed
- use a priority scheme

MULTIPLE INTERRUPTS



(a) Sequential interrupt processing



(b) Nested interrupt processing

MULTIPLE INTERRUPTS

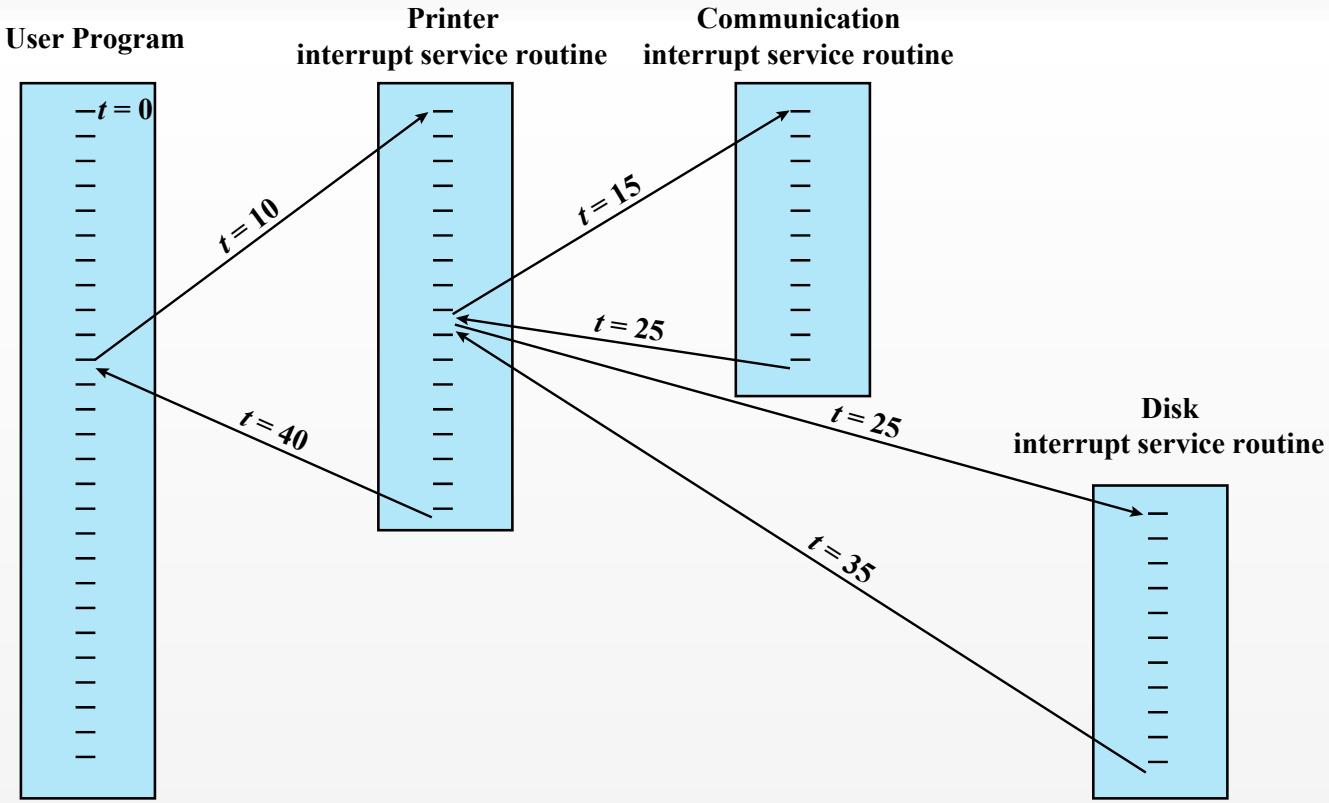


Figure 1.13 Example Time Sequence of Multiple Interrupts

THE MEMORY HIERARCHY

The design constraints on a computer's memory can be summed up by three questions:

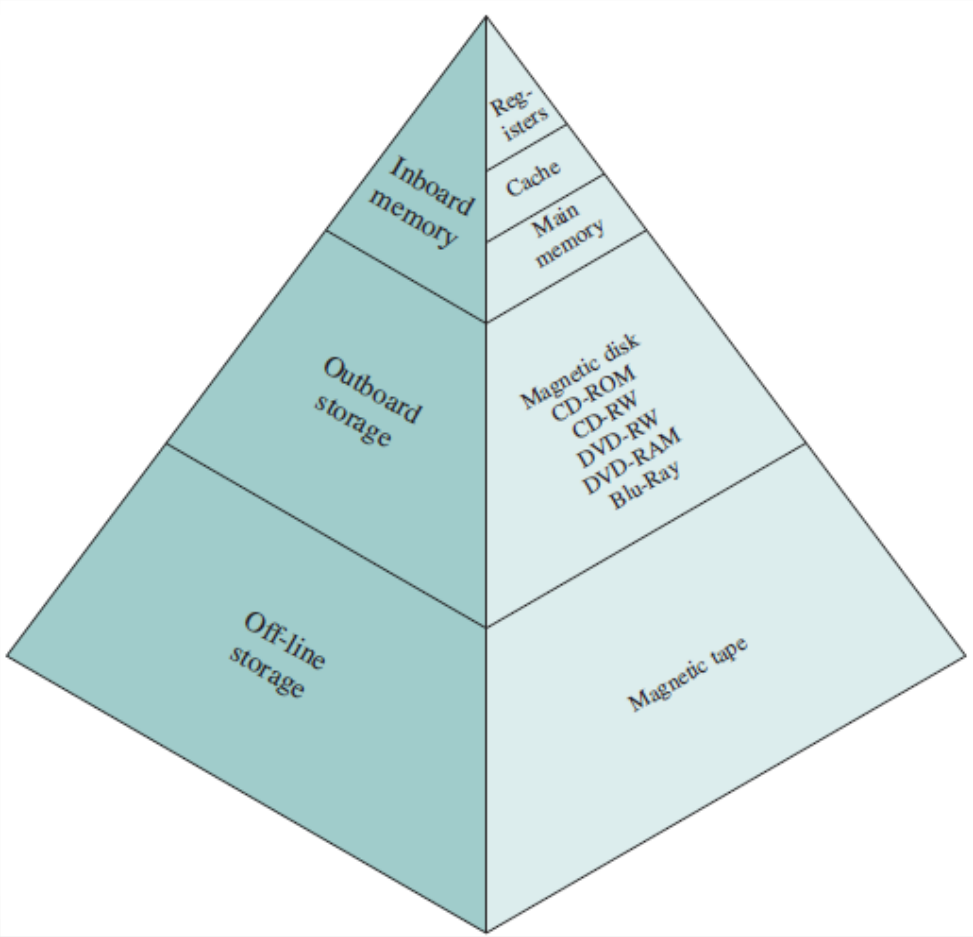
How much? . If the capacity is there, applications will likely be developed to use it.

How fast? easier to answer. To achieve greatest performance, the memory must be able to keep up with the processor.

How expensive? For a practical system, the cost of memory must be reasonable in relationship to other components.

THE MEMORY HIERARCHY

- there is a trade-off among the three key characteristics of memory: namely, **capacity, access time,** and **cost.**
  - **Faster access time, greater cost per bit**
  - **Greater capacity, smaller cost per bit**
  - **Greater capacity, slower access speed**
- The way out of this dilemma is to not rely on a single memory component or technology, but to employ a **memory hierarchy**.
- A typical hierarchy is shown. As one goes down the hierarchy, the following occur:
  - **Decreasing cost per bit**
  - **Increasing capacity**
  - **Increasing access time**
  - **Decreasing frequency of access to the memory by the processor**
- **The key to the success of this organization is the decreasing frequency of access at lower levels.**



LOCALITY OF REFERENCE PRINCIPLE

- During the course of execution of a program, memory references by the processor, for **both instructions and data, tend to cluster.**
- Programs typically contain a number of iterative loops and subroutines. Once a loop or subroutine is entered, there are repeated references to a small set of instructions.
- Similarly, operations on tables and arrays involve access to a clustered set of data bytes. Over a long period of time, the clusters in use change, but over a short period of time, the processor is primarily working with fixed clusters of memory references.

SECONDARY MEMORY

- **Also referred to as auxiliary memory**
  - **external**
  - **nonvolatile**
  - **used to store program and data files**
  - **Example is Hard Disk**

- A hard disk is also used to provide an extension to main memory known as **virtual memory**
- Additional levels can be effectively added to the hierarchy in software. For example, a portion of main memory can be used as a buffer to temporarily hold data that are to be read out to disk. Such a technique, sometimes referred to as a **disk cache**

CACHE MEMORY

- Invisible to the OS
- Interacts with other memory management hardware
- Processor must access memory at least once per instruction cycle
- Processor execution is limited by memory cycle time
- Exploit the principle of locality with a small, fast memory

CACHE MEMORY