# UNIT 1. INTRODUCTION: COMPUTER & OPERATING SYSTEMS

## 1.1 BASIC ELEMENTS

A computer consists of processor, memory and I/O components with one or more modules of each type. These components are interconnected to ease job of computer. Thus, there are four structural elements:

- **Processor:** It controls the operation of the computer and performs its data processing functions. When there is only one processor, it is called as Central Processing Unit (CPU).
- **Main Memory:** It stores data and programs. But, the contents of memory are lost when the computer shuts down. Whereas, the contents of disk memory are retained.
- **I/O Modules:** Move data between the computer and its external environment. The external environment consists of a variety of devices including secondary memory devices, communications equipment and terminals.
- **System Bus:** This provides communication among processors, main memory and I/O modules.

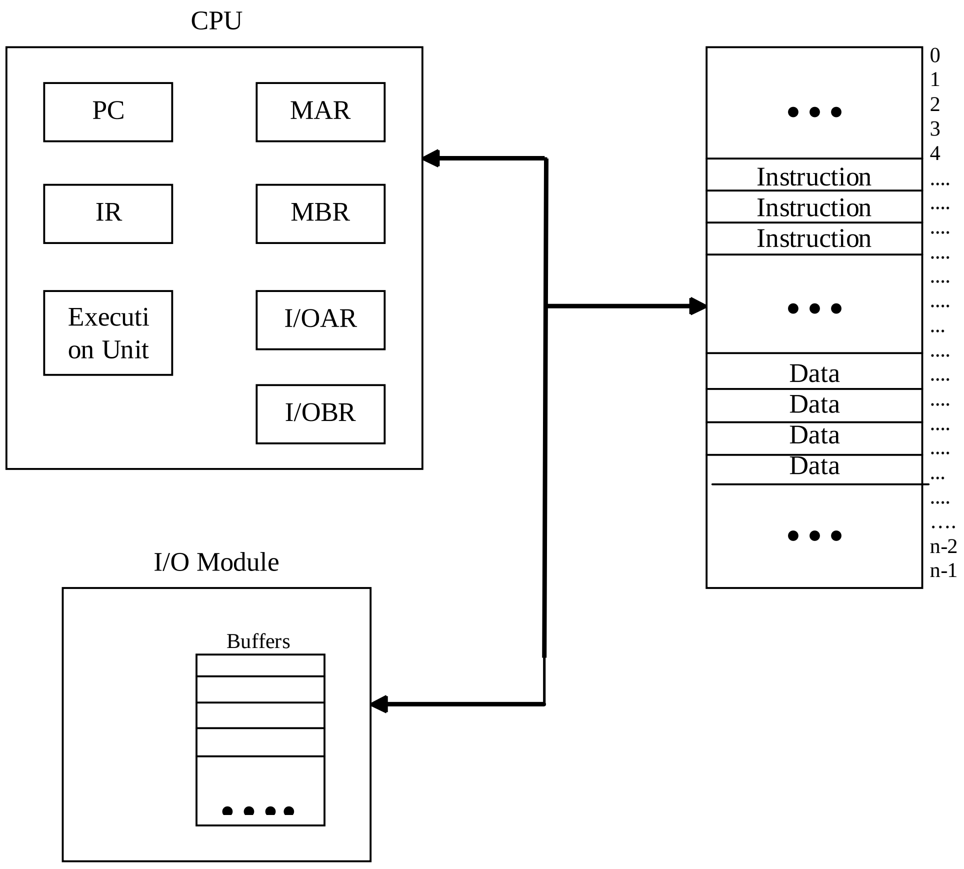


Figure 1.1 Computer Components: Top Level View

The top-level components of computer are shown in Figure 1.1. One of the functionality of a processor is to exchange data with memory. For this purpose, the following registers are used:

- **Memory Address Register (MAR):** specifies the address in memory for the next read or write.
- **Memory Buffer Register (MBR):** contains data to be written into memory or it receives the data read from memory.
- **I/O Address Register (I/OAR):** indicates particular I/O device
- **I/O Buffer Register:** used to exchange data between an I/O module and the processor.

A memory module contains a set of locations – which are sequentially numbered addresses. Each location contains a bit pattern that can be interpreted as an instruction or data. An I/O module transfers data from external devices to processor and memory and vice versa. It contains temporary buffers for holding data till they are sent.

## 1.2 PROCESSOR REGISTERS

A processor includes a set of registers that provide memory. But, this memory is faster and smaller than the main memory. These registers can be segregated into two types based on their functionalities as discussed in the following sections.

### 1.2.1 User – visible Registers

These registers enable the assembly language programmer to minimize the main memory references by optimizing register use. Higher level languages have an optimizing compiler which will make a choice between registers and main memory to store variables. Some languages like C allow the programmers to decide which variable has to be stored in register.

A user visible register is generally available to all programs. Types of registers that are available are: data, address and condition code registers.

- **Data Registers:** They can be assigned to different types of functions by the programmer. Sometimes, these are general purpose and can be used with any machine instruction that performs operations on data. Still, there are some restrictions like – few registers are used for floating-point operations and few are only for integers.

- **Address Registers:** These registers contain main memory addresses of data and instructions. They may be of general purpose or may be used for a particular way of addressing memory. Few examples are as given below:
    - **Index Registers:** Indexed addressing is a common mode of addressing which involves adding and index to a base value to get the effective address.
    - **Segment Pointer:** In segmented addressing, a memory is divided into segments (a variable-length block of words). In this mode of addressing, a register is used to hold the base address of the segment.

- **Stack Pointer:** If there is a user-visible stack addressing, then there is a register pointing to the top of the stack. This allows push and pop operations on instructions stored in the stack.

### 1.2.2 Control and Status Registers

The registers used by the processor to control its operation are called as Control and Status registers. This registers are also used for controlling the execution of programs. Most of such registers are not visible to the user. Along with MAR, MBR, I/OAR and I/OBR discussed earlier, following registers are also needed for an instruction to execute:

- **Program Counter:** that contains the address of next instruction to be fetched.
- **Instruction Register(IR):** contains the instruction most recently fetched.

All processor designs also include a register or set of registers, known as ***program status word (PSW)***. It contains condition codes and status information like interrupt enable/disable bit and kernel/user mode bit.

***Condition codes*** (also known as ***flags***) are bits set by the processor hardware as the result of operations. For example, an arithmetic operation may produce a positive, negative, zero or overflow result. Condition code bits are collected into one or more registers. And, they are the part of a control register. These bits only can be read to know the feedback of the instruction execution, but they can't be altered.

## 1.3 INSTRUCTION EXECUTION

A program to be executed contains a set of instructions stored in the memory. The processor takes two steps for processing an instruction:

- Read( or fetch) instructions from the memory one at a time
- Execute each instruction

These two steps are referred as ***fetch stage*** and ***execute stage*** respectively. One instruction requires both of these steps for its execution and such a processing is called as ***instruction cycle*** as shown in Figure 1.2. The program halts its execution only if the processor is turned off, some error occurs or there is an instruction in the program to terminate.

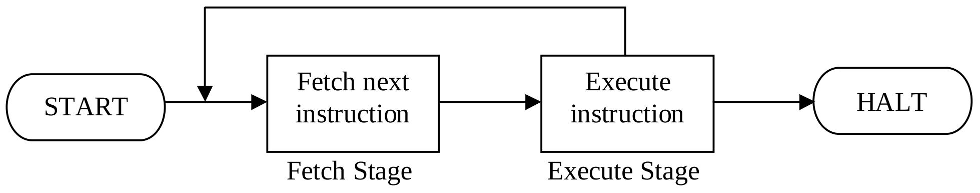


Figure 1.2 Basic Instruction Cycle

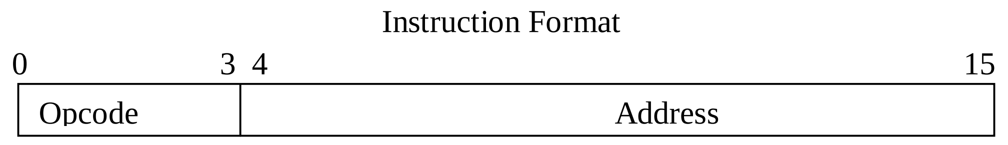
### 1.3.1 Instruction Fetch and Execute

At the beginning of every instruction cycle, the processor fetches an instruction from the memory. The program counter (PC) holds the address of next instruction to be fetched. And, the PC will be incremented whenever the processor fetches the instruction. For example, assume that the current value of PC is 300. When the processor fetches next instruction, PC will be incremented to 301. However, this logic may change in case of possible conditional statements of the program.

Later, the fetched instruction is loaded into the instruction register (IR). The instruction contains bits, and these bits inform the processor about the action to be taken. The processor understands these bits and performs the required action. Generally, this entire process of instruction execution is segregated into four categories as given below:

- **Processor-memory:** Data may be transferred from processor to memory and vice-versa.
- **Processor-I/O:** Data may be transferred to/from a peripheral device by transferring between the processor and I/O module.
- **Data Processing:** Processor may perform arithmetic/logical operations on data.
- **Control:** An instruction may specify that the sequence of execution be altered. For example, the processor may fetch the instruction from the location 149, which indicates the next address to be fetched should be 182. So, now the program counter is set to 182, instead of 150.

With the help of example, we will discuss these points in details now. Consider the processor with characteristics as shown in Figure 1.3.



CPU Registers:
Program Counter (PC) – Address of instruction
Instruction Register (IR) – Instruction being executed
Accumulator (AC) – Temporary storage

Partial list of opcodes:
0001 – Load AC from memory
0010 – Store AC to memory
0101 – Add to AC from memory

Figure 1.3 Characteristics of a processor

The processor contains single data register called accumulator (AC). Both data and instructions are 16 bit long. The instruction format allows 4 bits for the opcode. So, 16 different opcodes are possible (2^4=16). It will be a single hexadecimal digit. Remaining 12 bits are used for the address in the form of 3 hexadecimal digits. Consider the illustration of program execution (just a portion) shown in Figure 1.4. The program segment is to add the contents at the address 940 and the contents at the address 941 and then storing the result in the address 941. This is something similar to a programming statement x = y+x.

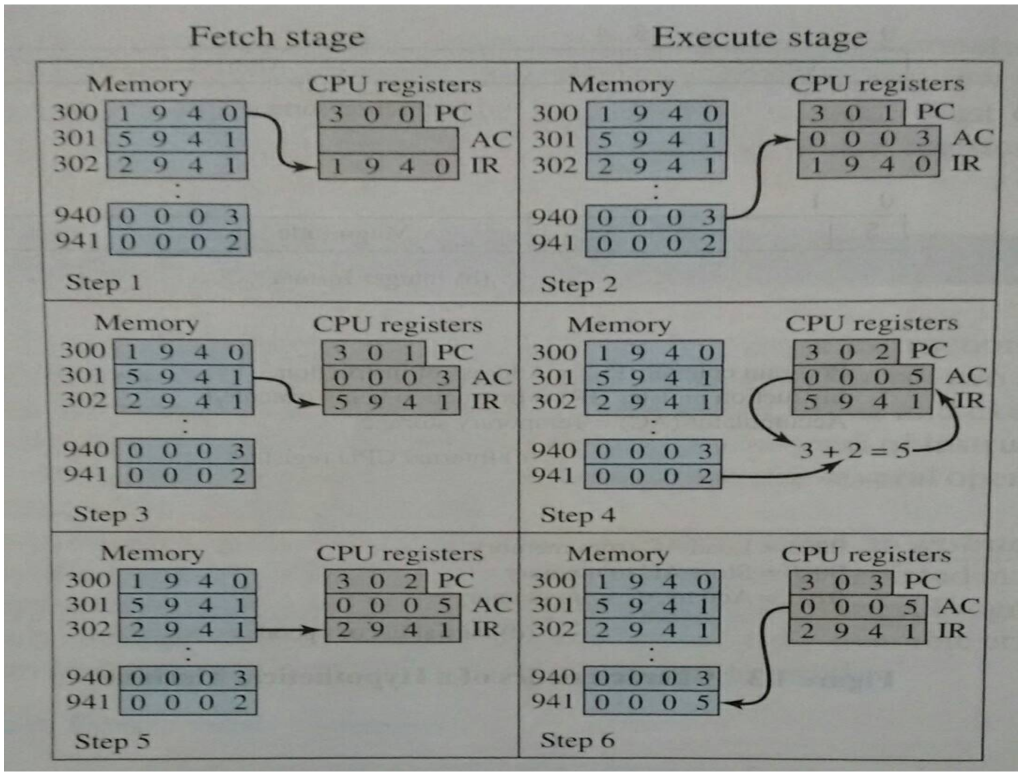


Figure 1.4 Example of program execution

For doing this job, three instruction cycles (3 fetch, 3 execute) are required as given below:

- **Fetch:** The PC contains the address of first instruction, that is 300. The instruction is 1940. Here, 1 indicates opcode and 940 is the memory address. The instruction 1940 is loaded into IR and PC is incremented to 301.
- **Execute:** Since the opcode 1 (or 0001) is for loading the AC from memory, the content of the address 940 is loaded into AC. Hence, now AC contains 0003.
- **Fetch:** Now, the next instruction (5941) is fetched from the location 301 and PC is incremented to 302.
- **Execute:** The opcode 5 (0101) indicates adding AC from memory. So, the content of the location 941 is added to the contents of AC. (0003 + 0002 =0005)
- **Fetch:** The next instruction (2941) from the address 302 is fetched and the PC is incremented to 303.
- **Execute:** The opcode 2 (0010) indicates storing AC to memory. Hence the value 0005 is loaded into the memory address 941.

### 1.3.2 I/O Function

Data can be exchanged directly between an I/O module and the processor. That is, the processor can directly read/write data from/to I/O module, not necessarily from the memory. Here, the processor identifies a specific device that is controlled by a particular I/O module.

In some situations, it is better to allow I/O exchanges to occur directly with main memory to relieve the processor from I/O task. In such cases, the processor should grant the authority to I/O module to read/write to memory. Hence, processor is not tied up with I/O operations. Now, I/O module will issue read/write commands directly to the memory. This operation is known as ***direct memory access (DMA)***.

## 1.4 INTERRUPTS

The normal sequence of the processor may be interrupted by other modules like I/O, memory etc. Table 1.1 gives the list of common interrupts.

Table 1.1 Classes of Interrupts

| Class | Description |
|---|---|
| Program | Generated as a result of an instruction execution. For example, arithmetic overflow, division by zero etc. |
| Timer | Generated by timer within the processor. It allows OS to perform certain functions on regular basis. |
| I/O | Generated by I/O controller to indicate any error or normal completion of an operation. |
| Hardware Failure | Generated by failure like power failure or memory parity error. |

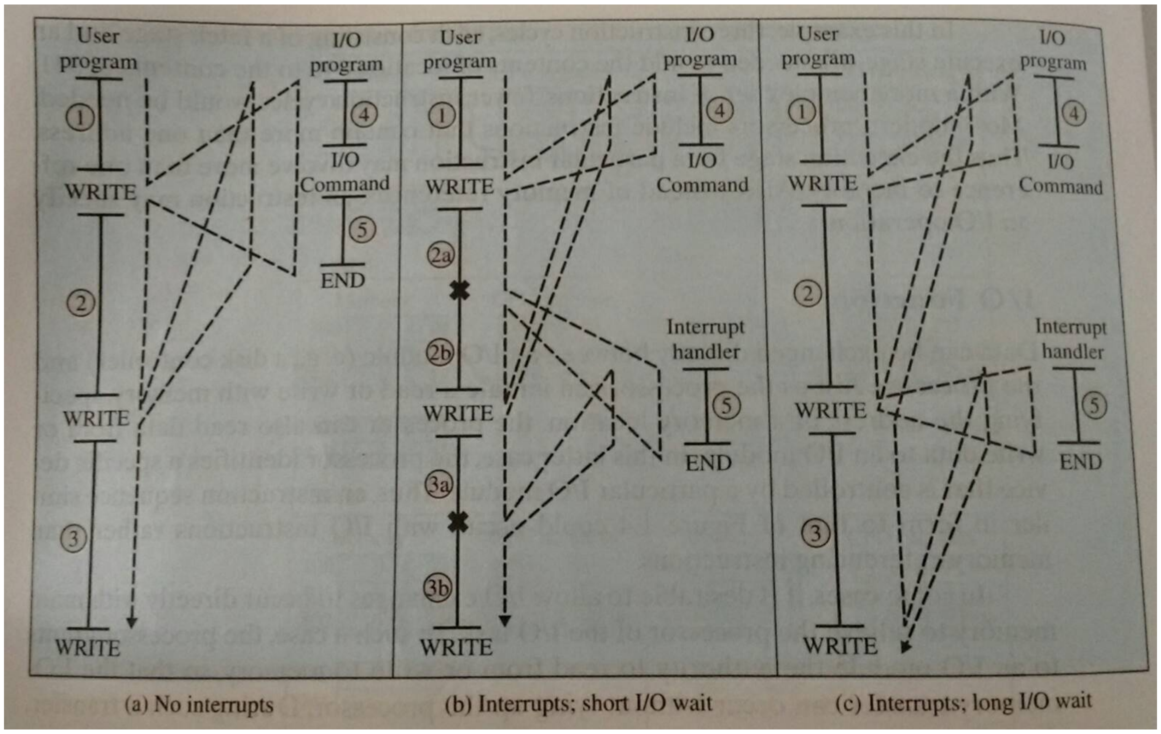


Figure1.5 Program flow of control with and without interrupts

The aim of interrupts is to improve the processor utilization. For example, most I/O devices are slower than the processor. If the processor gives the instruction for WRITE something on I/O devices, the I/O unit takes two steps for the job –

- I/O program may copy the data to be written into the buffer etc. and prepare for the actual I/O operation.
- The actual I/O command has to be executed.

Without interrupts, the processor would sit idle while the I/O unit is preparing (the first step) itself for the job. But, in case of interrupts, the processor just gives intimation to the I/O unit first. While I/O unit prepares itself, the processor would continue to execute the next instructions in the program. When I/O unit is ready, in between, it will do the actual I/O command and come back to normal flow of execution. Refer Figure 1.5 for understanding this concept.

### 1.4.1 Interrupts and the Instruction Cycle

Whenever interrupts are introduced in the system, the processor gives information to the I/O unit and without waiting for I/O operation to complete; it will continue to execute next instruction. When the external device is ready to accept more data from the processor, the I/O module sends an ***interrupt request*** signal to the processor. Now, the processor suspends the current operation of the program and responds to a routine (or a function) of I/O device, which is called as ***interrupt handler***. When the interrupt processing is completed, the processor resumes the execution. To allow interrupts, an ***interrupt stage*** is added along with fetch stage and execute stage as shown in Figure 1.6.

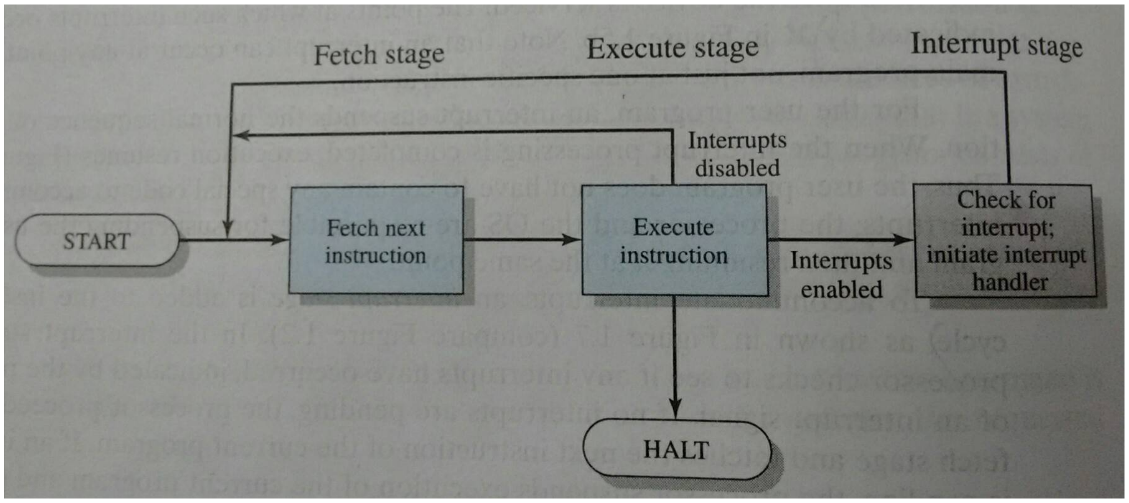


Figure 1.6 Instruction Cycle with interrupts

In the interrupt stage, the processor checks for any possible interrupt signal. If no interrupt is pending, it will go the fetch stage. If an interrupt signal is there, the processor suspends current execution and executes interrupt handler routine. The interrupt handler routine is a part of OS, which identifies nature of interrupt and performs necessary action. After completing interrupt handler routine, the processor resumes the program execution from the point where it was suspended.

It is understood that some overhead is involved in this process. Extra instructions have to be executed in the interrupt handler to determine type of interrupt, to decide the appropriate action etc. But, instead of processor sitting idle for I/O operation and wasting huge amount of time, the concepts of interrupts are found to be efficient.

### 1.4.2 Interrupt Processing
An interrupt triggers many events in the processor hardware and software. The Figure 1.7 shows the sequence of these events and is explained below:

i. The I/O device (or any other interrupt) issues an interrupt signal to the processor.
ii. The processor finishes the execution of current instruction.
iii. The processor checks for the interrupt signal and send the acknowledgement to the I/O device. This acknowledgement helps the device to clear the interrupt signal.
iv. Now, the processor has to transfer the control to interrupt routine. Before that, it saves the current status of PC and PSW (Program Status Word) in the control stack. This will help the processor to resume its execution after finishing the interrupt routine.
v. Then, the processor loads the entry location of interrupt handling routine into the PC.

Now, the processor has to go for next instruction cycle by fetching the address at PC. But, PC now contains address of interrupt routine (as per step (v)) and hence, the following operations will be carried out.

vi. The PC and PSW relating to the interrupted program have been saved on the control stack. Now, the contents of all registers are also pushed into the control stack. Thus, the top of the stack contains the address of interrupt routine.
vii. The interrupt handler will now process the interrupt.
viii. When interrupt processing is complete, the saved register values are retrieved from the stack and stored back into the registers.
ix. Finally, PC and PSW are loaded with their old values from the stack. Hence, the next instruction of the program will be executed.

### 1.4.3 Multiple Interrupts
It is possible to have multiple interrupts in a single program. One or more interrupt can occur during another interrupt is being processed. For example, an input may be taken while printing the data onto the printer. Each time, the printer finishes its job, an interrupt will occur. There are two approaches to deal with multiple interrupts as explained below (Refer Figure 1.8).

- **Disable the interrupts while one interrupt is being processed:** Here, the processor ignores any new interrupt signal when another interrupt is in progress. The processor keeps such signals as pending, and considers when the previous interrupt routine is completed. Hence, all the interrupts will be processed sequentially.

  But, in this approach, the priorities or inter-dependencies between the interrupt routines are not considered. Hence, time-critical needs cannot be satisfied.

- **Define priorities for interrupts:** Here, a higher priority interrupt will pause the lower priority interrupt which is in execution. Hence, a nested interrupt processing will be achieved based on the priority.

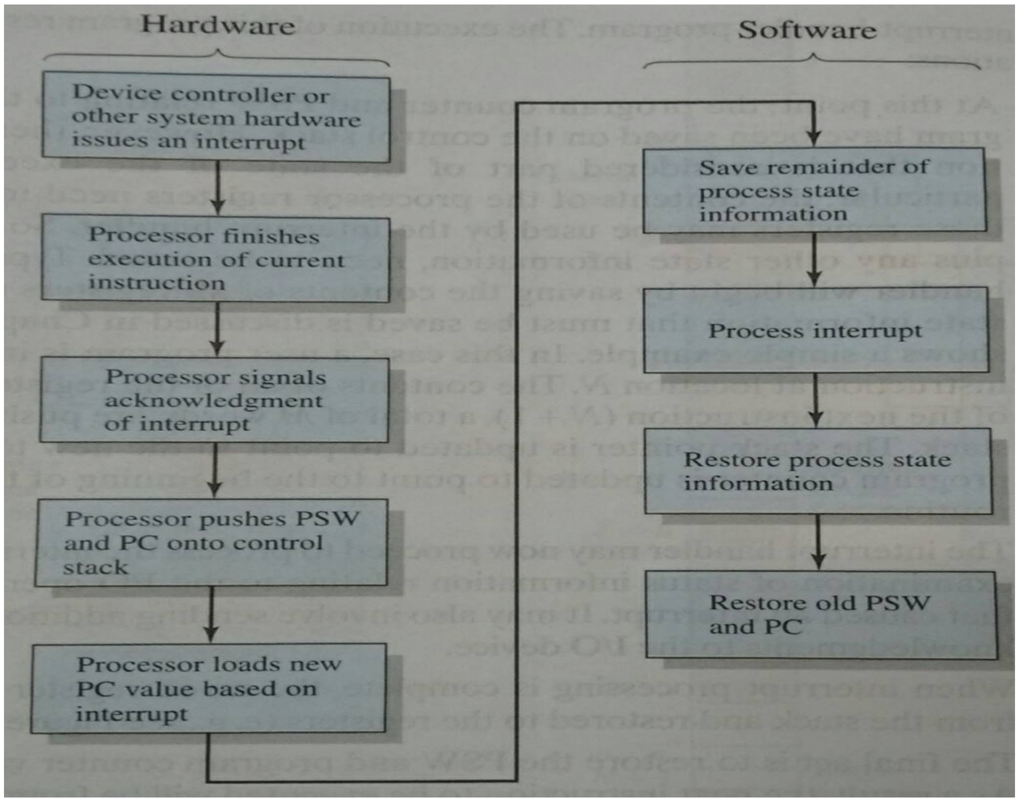
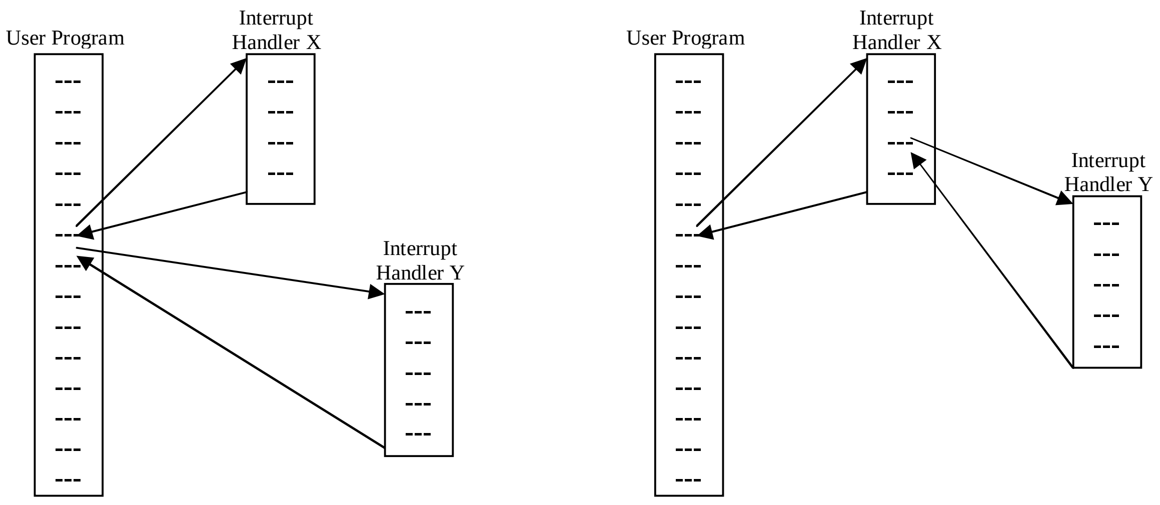

Hardware
Software
Device controller or other system hardware issues an interrupt
Processor finishes execution of current instruction
Processor signals acknowledgment of interrupt
Processor pushes PSW and PC onto control stack
Processor loads new PC value based on interrupt
Save remainder of process state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

Figure 1.7 Simple interrupt processing



(a) Sequential Interrupt Processing (b) Nested Interrupt Processing

Figure 1.8 Transfer of control during multiple interrupts

### 1.4.4 Multiprogramming

Even though interrupts are used, in most of the situations, the processor is not being used efficiently. For example, if there is a long I/O wait, then time taken for I/O is more than the actual user code. So, processor would sit idle. To solve this problem, multiple programs may be made to execute. Thus, when one program is busy with I/O, the other program's instructions may be getting executed; and vice-versa. When the processor is dealing with many programs, the sequence in which the programs are executed will depend on their relative priority. In this case, when one interrupt routine is completed, the processor may not come back to the user program instructions, but it may consider interrupt of another program based on the priority.

## 1.5 THE MEMORY HIERARCHY

The constraints on the design of computer's memory depend on three key points viz.

- Capacity (how much is the size?)
- Access time (how fast it can be accessed?)
- Cost (how expensive it is?)

In general, the relationship between these three points will be as below –

- Faster access time, greater the cost per bit
- Greater capacity, smaller cost per bit
- Greater capacity, slower access speed

Thus, the designers of computer memory face the dilemma on these points. He/she has to optimize and balance the constraints.

To solve this problem, **memory hierarchy** has to be used instead of relying on a single memory component. A typical memory hierarchy would be as given in Figure 1.9. In this hierarchy, from top to bottom, the following occur:

(i) Decreasing cost per bit
(ii) Increasing capacity
(iii) Increasing access time
(iv) Decreasing frequency of access to the memory by the processor

Hence, the smaller, expensive, faster memories are supplemented by larger, cheaper, slower memories. The levels of memory hierarchy are explained hereunder.

- **Registers:** Generally every processor contains a few dozen or hundreds of registers which are faster, smaller but expensive.
- **Cache Memory:** It is not usually visible to the programmer, but visible only to the processor. It is used for the movement of data between main memory and processor registers.
- **Main Memory:** It is the principal internal memory system of the computer. Each location in the main memory has a unique address. Most of the machine instructions refer to one or more main memory addresses. Main memory is usually extended with a higher speed, smaller cache.
- **Secondary/Auxiliary Memory:** The next two levels of the hierarchy falls into this category. The data are stored permanently on external storage devices like hard disk,

removable devices like CD, DVD, tape etc. Programmer can see such data in the form of folders and files.

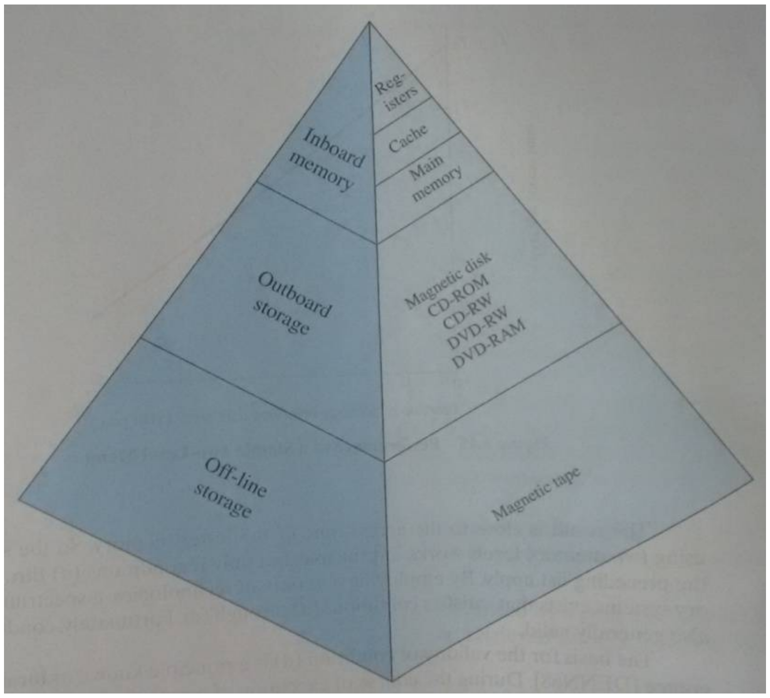


Figure 1.9 Memory Hierarchy

## 1.6 CACHE MEMORY

Since cache memory plays very important role in the performance of the processor and managing the memory hardware, it is being discussed in detail here.

### 1.6.1 Motivation

On every instruction cycle, the processor fetches the memory at least once. If the instruction contains operands or it needs to store some data, then every fetch may need to access memory more than once. Thus, the processor speed is always restricted by the memory cycle time. And, in almost all computers, the processor speed is much higher than that of memory cycle speed. Hence, an intermediate repository of instructions is thought of to keep a portion of memory that needs to be executed by the processor. Such a small and fast memory between the processor and main memory is called as cache.

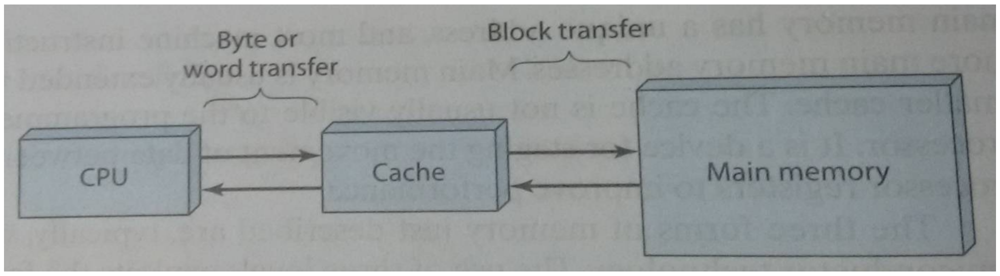


Figure 1.10 Cache and Main Memory

### 1.6.2 Cache Principles

The working of cache memory is depicted in Figure 1.10. The cache size will be considerably smaller than then main memory. It contains a copy of some portion of main memory. When the processor tries to read a byte/word from the memory, the cache is checked first. If that byte is available in the cache, it is delivered to the processor. If not, a block of main memory containing that byte will be stored into the cache and then it is delivered to the processor. Figure 1.11 shows the process of read operation using cache.

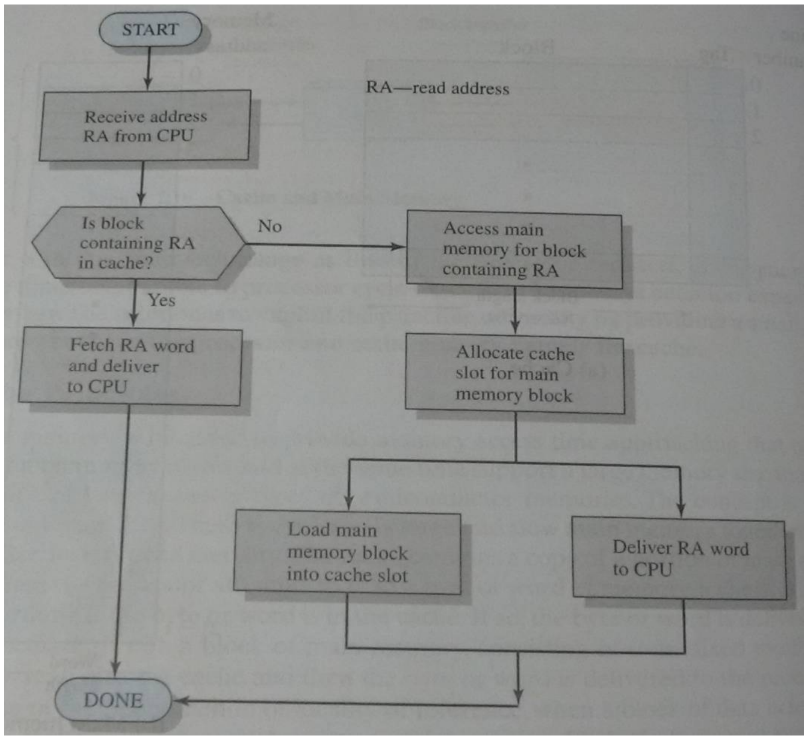

START
RA—read address
Receive address RA from CPU
Is block containing RA in cache?
No
Access main memory for block containing RA
Yes
Fetch RA word and deliver to CPU
Allocate cache slot for main memory block
Load main memory block into cache slot
Deliver RA word to CPU
DONE

Figure 1.11 Read operation using cache

### 1.6.3 Cache Design

The design of a cache memory has to consider following aspects:

- **Cache size:** Small size caches have significant impact on the performance of the processor.
- **Block size:** It indicates the amount of data exchanged between cache and main memory. Optimum selection of this size is essential. Because, if the block size is too small, then it cannot hold many instructions and hence the main memory has to be hit more number of times. Whereas, if the block size is too large, then it becomes almost like a main memory and the very usage of cache will be void.
- **Mapping function:** This function determines the location in the cache to be occupied by the block. It has two constraints: (i) while one block is read, another may need to be replaced. (ii) As mapping function becomes more flexible, the design circuitry becomes complex.
- **Replacement algorithm:** This algorithm decides which block has to be replaced when a new block is loaded into the cache. And the design of this algorithm should

work within the constraints of mapping function. In most of the cases, least-recently-used (LRU) method is applied.
- **Write policy:** If the contents of the block in the cache are modified, the same has to be written inside the main memory. The write policy indicates when the memory write operation has to take place.

## 1.7 I/O COMMUNICATION TECHNIQUES

I/O operations are possible using following three techniques:
- Programmed I/O
- Interrupt-driven I/O
- Direct Memory Access (DMA)

Each of these techniques is explained here and the diagrammatic representation is given in Figure 1.12.

### 1.7.1 Programmed I/O

When the processor is executing a program and encounters an I/O instruction, then it will inform I/O module and executes that instruction. In case of programmed I/O, the I/O module performs the task but do not interrupt the processor about the completion of the task. Hence, the processor must periodically keep checking the I/O module for the completion of the task.

Thus, the processor is responsible for extracting/storing data from/to the main memory. So, the instruction set includes the I/O instructions in the following categories:
- **Control:** activates an external device and informs the action to be taken.
- **Status:** used to test various status conditions associated with I/O module.
- **Transfer:** to read/write data between processor registers and external devices.

Note that, the technique of programmed I/O is time-consuming and keeps the processor busy unnecessarily.

### 1.7.2 Interrupt-Driven I/O

As it is observed, the programmed I/O technique will degrade the performance of the processor. An alternative way is to provide interrupt-driven I/O. In this technique, the processor will issue an I/O command to the I/O module and then continue its regular instruction execution. When the I/O module is ready, it will interrupt the processor. Then the processor will execute the requested task and then resume its former processing.

### 1.7.3 Direct Memory Access

Though interrupt-driven I/O is efficient than the programmed I/O, it requires active participation of the processor for transferring data between memory and I/O module. Hence, both of these techniques have following drawbacks:
- I/O transfer rate is limited
- Processor is tied up in managing I/O transfer

To avoid these problems, direct memory access (DMA) is proposed. It can be put as a separate module on the system bus or as a part of I/O module. In this technique, when the processor has read/write data, it issues a command to DMA by sending following information:

- Whether a read or write is requested
- Address of the I/O device involved
- Starting location in memory to read/write data
- Number of words to be read/written

Then, the processor continues its work. Now the job has been delegated to DMA module. The DMA module will transfer the entire data from the memory and then interrupts the processor. Thus, the processor is involved only at the beginning and ending of the data transfer.

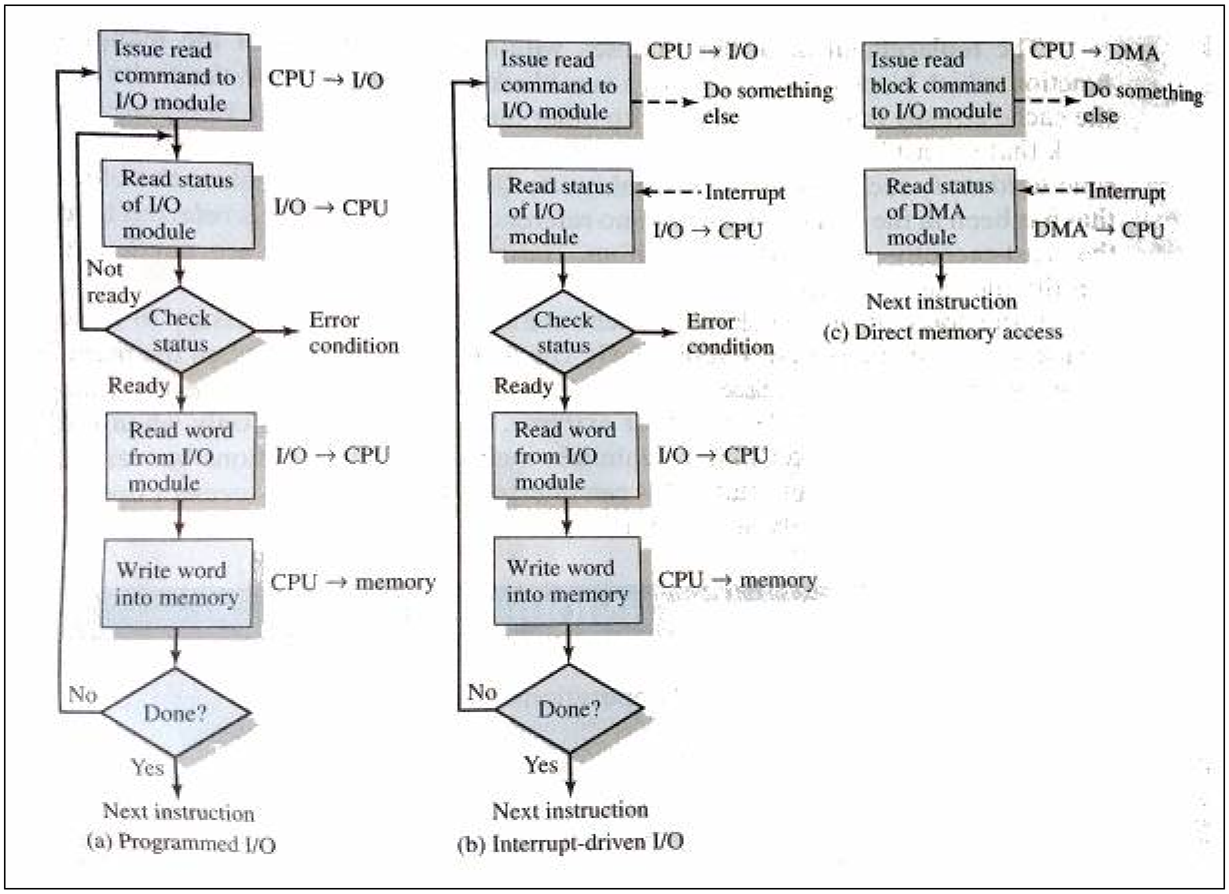


Figure 1.12 Techniques of I/O communication

## 1.8 INTRODUCTION TO OPERATING SYSTEM

An operating system is a program that manages the computer hardware. It also provides a basis for application programs and acts as an intermediary between a user of a computer and the computer hardware.

The design of OS is depending on the purpose for which it is being used. For example,

- Mainframe OS are designed to optimize the utilization of hardware

- Personal computer OS are designed to support complex games, business applications etc.
- Handheld computer (like mobiles, tablets etc) OS provides user-friendly interface and environment to execute programs/applications.

Hence, few OS may be convenient, few may be efficient and some may be combination of both.

A computer system can be divided into four components as shown in Figure 1.13. They are:

- **Hardware** : Consists of central processing unit (CPU), memory and I/O devices and provides the basic computing resources
- **Operating system:** Controls and coordinates the use of hardware among various application software for different users.
- **Application programs:** defines the ways in which the computing resources are used to solve the problems of users. For example, word processors, spreadsheets, compilers, browsers etc.
- **Users:** users of the computer.

Thus, we can say that OS is like a government. It does not perform any useful function by itself, but provides environment within which other programs can do useful works.

OS can be explored from two viewpoints and are explained below:

- User View
- System View

### 1.8.1 User View

The user's view of the computer varies according to the interface being used. The design on OS varies depending on type of the user or work to be carried out as explained below:

- **User of PC:** OS for personal computers is aimed at maximizing the work that the user is performing. Hence, the OS design is mostly for the ease of use rather than the performance. And, resource utilization is not taken into consideration.
- **User of Mainframes:** Some users make use of terminals connected to a mainframe or minicomputer. There will be other people accessing the same computer through other terminals. These users share the resources and information. Hence, the OS on such systems is designed to maximize the resource utilization. Thus, efficient sharing of CPU time, memory and I/O are assured for every user.
- **User of Workstations:** Sometimes, computers are connected to networks of workstations and servers. The user working on such workstations may have their own resources and they may also share the resources with other servers. Hence, OS for such situation is designed to optimize between individual usability and resource utilization.
- **User of Handheld Computers:** Handheld devices like mobiles, tablets are common nowadays. Because of power, speed and interface limitations, they may perform relatively less operations. So, their OS are designed for individual usability by keeping battery life in mind.

Some computers have little or no user view. For example, embedded computers in home devices and automobiles may have numeric keypads and may turn indicator lights on or off to show status, but they and their operating systems are designed primarily to run without user intervention.

### 1.8.2 System View

From the computer's point of view, the OS is the program closely involved with the hardware. In this context, we can view an OS as a **resource allocator.** A computer system has many resources that may be required to solve a problem: CPU time, memory space, file-storage space, I/O devices, and so on. The OS acts as the manager of these resources by allocating the resources to the programs and users efficiently.

Another view of OS focuses on controlling various I/O devices and user programs. As a control program, OS manages the execution of user programs to prevent errors and improper use of computer.

The aim of any computer system is to execute user programs to solve user's problems. As computer hardware alone is not easy to use, application programs/softwares have been created for solving the problems. Such programs require certain common operations like controlling I/O devices etc. The common functions of controlling and allocating resources are brought together into one piece of software known as Operating System.

The widely accepted definition of OS goes like this: ***OS is the program running all times on the computer (also called as kernel), with all other programs being treated as application programs.***

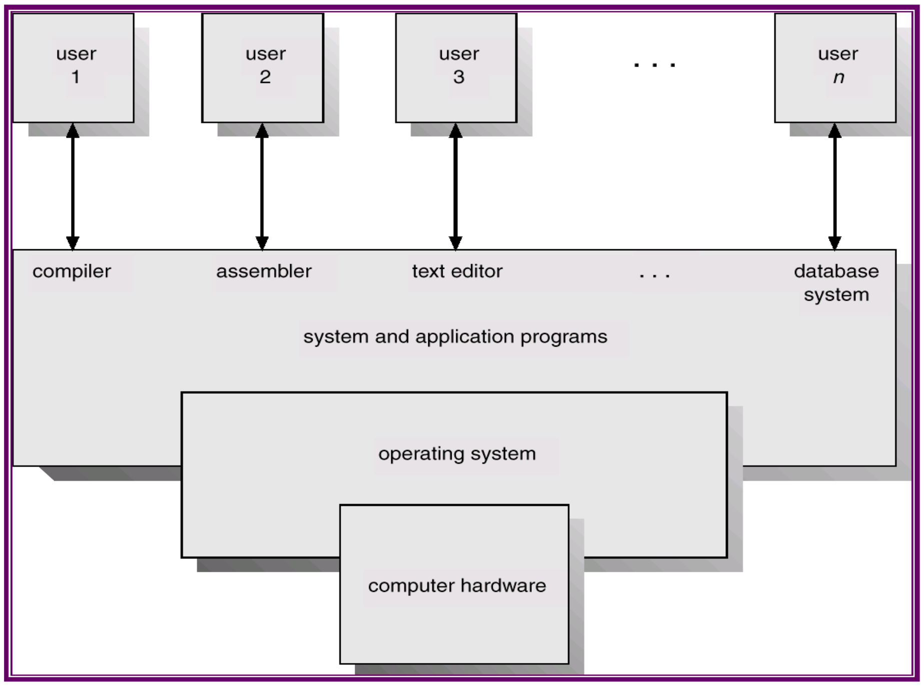


Figure 1.13 Abstract view of components of a computer system

### 1.8.3 System Goals

It is easier to define an operating system by what it *does* than by what it *is*. The primary goal of some operating system is *convenience for the user.* Operating systems exist because they are supposed to make it easier to compute with them than without them. This view is particularly clear when you look at operating systems for small PCs.

The primary goal of other operating systems is *efficient* operation of the computer system. This is the case for large, shared, multi-user systems. These systems are expensive, so it is desirable to make them as efficient as possible.

These two goals - convenience and efficiency-are sometimes contradictory. In the past, efficiency was often more important than convenience. Thus, much of operating-system theory concentrates on optimal use of computing resources. Operating systems have also evolved over time. For example, UNIX started with a keyboard and printer as its interface, limiting how convenient it could be for the user. Over time, hardware changed, and UNIX was ported to new hardware with more user-friendly interfaces. Many **graphic** user interfaces (GUIs) were added, allowing UNIX to be more convenient for users while still concentrating on efficiency.

The designers of OS face many tradeoffs related to efficiency and convenience. Lot of continuous revision and updation is necessary. Still, the success of OS depends on its users. In past 50 years, OS evolved into different phases. And, OS and computer architecture have influenced each other. To facilitate the use of the hardware, researchers developed operating systems. Users of the operating systems then proposed changes in hardware design to simplify them.

## 1.9 MAINFRAME SYSTEMS

Mainframe computer systems were the first computers used to tackle many commercial and scientific applications. In this section, we trace the growth of mainframe systems.

### 1.9.1 Batch Systems

Early computers were huge machines and were run from a console. The common input devices were card readers and tape drives. The common output devices were line printers, tape drives, and card punches. The user did not interact directly with the computer systems. Rather, the user prepared a job and submitted it to the computer operator. Normally, a job would consist of the program, the data, and some control information about the nature of the job (control cards). The job was usually in the form of punch cards (a sample punch card is shown in Figure 1.15). After some time (may be minutes, hours, or days), the output appeared. The output consisted of the result of the program, as well as a dump of the final memory and register contents for debugging.

The operating system in these early computers was fairly simple. Its major task was to transfer control automatically from one job to the next. The operating system was always resident in memory as shown in Figure 1.14.

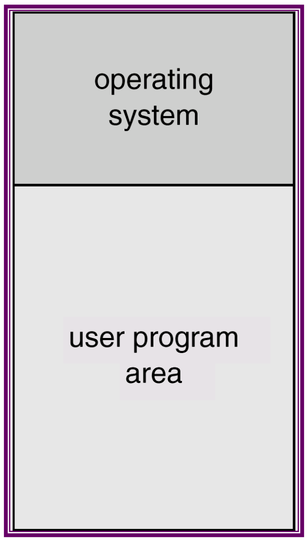


**Figure 1.14 Memory layout for simple batch system**

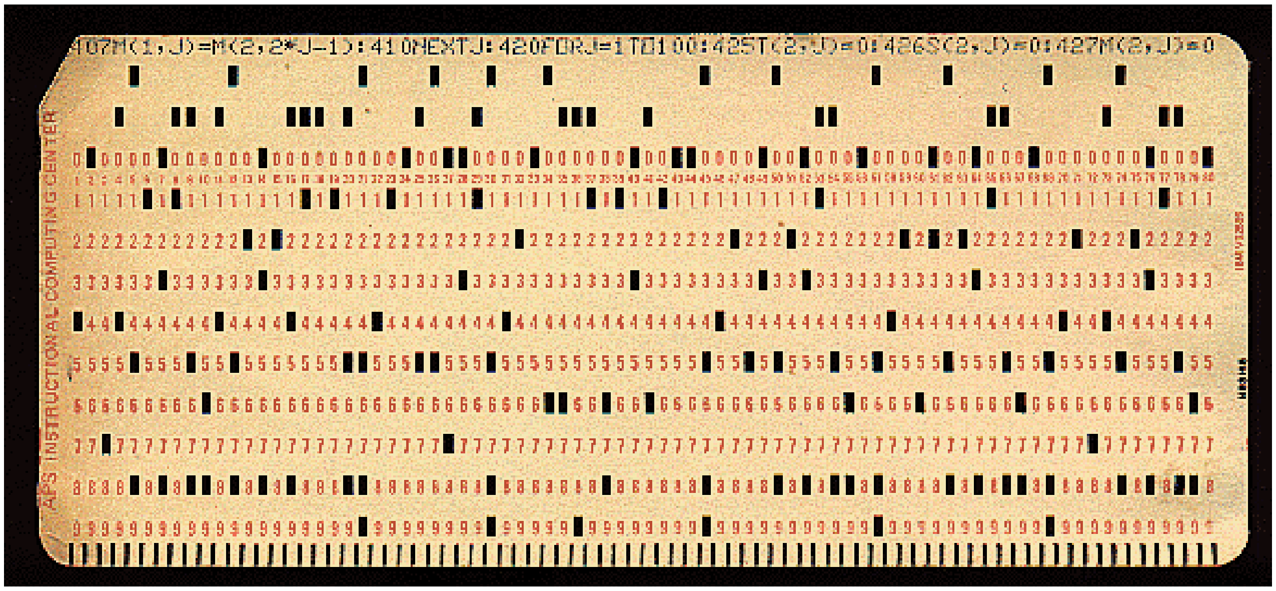

407M(1,J)=M(2,2*J-1):410NEXTJ:420FORJ=1TO100:425T(2,J)=0:426S(2,J)=0:427M(2,J)=0
APS INSTRUCTIONAL COMPUTING CENTER

Figure 1.15 A sample puchcard reader

The operators batched the jobs with similar needs together and ran them through the computer as a group. Later, the output from each job would be sent back to respective programmer. This would speed up the processing.

In this execution environment, the CPU is often idle, because the speed of I/O devices is much slower than process. Over time, improvements in technology and the introduction of disks resulted in faster I/O devices. However, CPU speeds increased to an even greater extent, so the problem was not only unresolved, but became worse. The introduction of disk technology allowed the operating system to keep all jobs on a disk, rather than in a serial card reader. With direct access to several jobs, the operating system could perform job scheduling, to use resources and perform tasks efficiently. Job scheduling is discussed later in detail.

### 1.9.2 Multiprogrammed Systems

The most important aspect of job scheduling is the ability to multiprogram. A single user cannot keep either the CPU or the I/O devices busy at all times. Multiprogramming increases CPU utilization by organizing jobs so that the CPU always has one job to execute.

In multiprogrammed systems, the OS keeps several jobs in memory simultaneously as shown in Figure 1.16. The OS picks and begins to execute one of the jobs in the memory. Eventually, the job may have to wait for some task, such as an I/O operation, to complete. In a non-multiprogrammed system, the CPU would sit idle. In a multiprogramming system, the OS simply switches to, and executes, another job. When that job needs to wait, the CPU is switched to another job, and so on. Eventually, the first job finishes waiting and gets the CPU back. As long as at least one job needs to execute, the CPU is never idle.

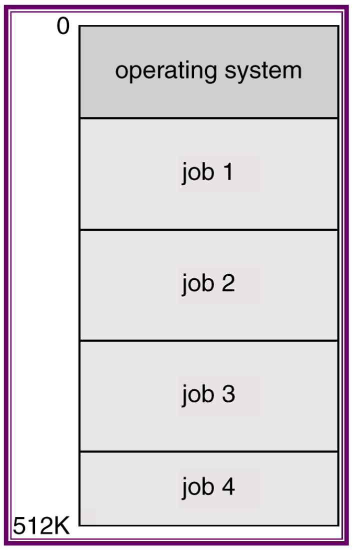


Figure 1.16 Memory layout for multiprogramming system

Multiprogramming creates the situation for OS to make certain decisions:

- There will be many jobs residing on the disk waiting for allocation of main memory for processing. The OS has to take decision of choosing one job among them (This is job scheduling).
- When the OS selects a job from the job pool, it loads that job into memory for execution. Having several programs in memory at the same time requires some form of memory management.
- In addition, if several jobs are ready to run at the same time, the system must choose among them. Making this decision is CPU scheduling.
- Finally, multiple jobs running concurrently require that their ability to affect one another be limited in all phases of the operating system, including process

scheduling, disk storage, and memory management. OS has to consider all these aspects.

### 1.9.3 Time-sharing Systems

Multiprogrammed, batched systems provided effective use of resources, but did not provide for user interaction with the computer system. Time sharing (or multitasking) is a logical extension of multiprogramming. Here, the CPU executes multiple jobs by switching among them, but the switches occur so frequently that the users can interact with each program while it is running. The user gives instructions to the operating system or to a program directly, using a keyboard or a mouse, and waits for immediate results. The response time should be short, typically within 1 second.

A time-shared OS allows many users to share the computer simultaneously. As the system switches rapidly from one user to the other, every user feels that the entire computer is dedicated to him/her.

A time-shared OS makes use of CPU scheduling and multiprogramming to provide a small portion of a time-shared computer to every user. Each user has at least one separate program in memory. ***A program loaded into memory and executing is commonly referred to as a process.*** A process normally executes only for a short time and waits to perform I/O. As I/O is depending on the speed of the user, which is very low compared to that of a system, the user gets enough time. Meanwhile, the OS takes up another process to execute.

In both time-sharing and multiprogrammed OS, several jobs must be kept simultaneously in memory. Hence, the OS should have memory management and protection. To obtain a reasonable response time, jobs may have to be swapped in and out of main memory to the disk. This is achieved by virtual memory, which is a technique that allows the execution of a job that may not be completely in memory. The main advantage of the virtual-memory scheme is that programs can be larger than physical memory. Further, it abstracts main memory into a large, uniform array of storage, separating logical memory as viewed by the user from physical memory. This arrangement frees programmers from concern over memory-storage limitations.

Time-sharing systems must also provide a file system. As the file system resides on a collection of disks, the disk management must be provided by OS. Also, time-sharing systems provide a mechanism for concurrent execution, which requires sophisticated CPU-scheduling schemes. To ensure orderly execution, the system must provide mechanisms for job synchronization and communication and it may ensure that jobs do not get stuck in a deadlock, forever waiting for one another.

## 1.10 DESKTOP SYSTEMS

Personal computers appeared in the 1970s. During their first decade, the CPUs in PCs lacked the features needed to protect an OS from user programs. Therefore, OS in PC were neither multiuser nor multitasking. However, the goals of these OS have changed with time. Instead of maximizing CPU and peripheral utilization, the systems opt for maximizing user convenience and responsiveness; for example, Microsoft Windows and Apple Macintosh. The MS-DOS from Microsoft has been superseded by multiple flavors of Microsoft Windows, and IBM has upgraded MS-DOS to the OS/2 multitasking system. The Apple Macintosh operating system has been ported to more advanced hardware, and now includes new features, such as virtual memory and multitasking. With the release of MacOS X, the core of OS is now based on Mach and FreeBSD UNIX for scalability, performance, and features, but it retains the same rich GUI. Linux, a UNIX-like operating system available for PCs, has also become popular recently.

OS for these computers have benefited in several ways from the development of OS for mainframes. Microcomputers were immediately able to adopt some of the technology developed for larger OS. On the other hand, the hardware costs for microcomputers are becoming less that an individual user can bear, and CPU utilization is no longer a prime concern. Thus, some of the design decisions made in OS for mainframes may not be appropriate for smaller systems.

Other design decisions still apply. For example, initially file protection was not necessary on a personal machine. However, these computers are now often tied into other computers over LAN or other Internet connections. When other computers and other users can access the files on a PC, file protection again becomes a necessary feature of the OS. The lack of such protection has made it easy for malicious programs to destroy data on systems such as MS-DOS and Macintosh. These programs may be self-replicating, and may spread rapidly via worm or virus mechanisms and disrupt entire companies or even worldwide networks. Advanced timesharing features such as protected memory and file permissions are not enough, on their own, to safeguard a system from attack. But, recent advancements in technology are safeguarding a system up to some extent.

## 1.11 MULTIPROCESSOR SYSTEMS

Nowadays, many systems are single-processor systems having only one main CPU. However, multiprocessor systems (also known as parallel systems or tightly coupled systems) are growing in importance. Such systems have more than one processor in close communication, sharing the computer bus, the clock, and sometimes memory and peripheral devices.

Multiprocessor systems have three main advantages:

- **Increased throughput:** By increasing the number of processors, we hope to get more work done in less time. The speed-up ratio with N processors is not N; rather, it is less than N. When multiple processors cooperate on a task, a certain amount of overhead is incurred in keeping all the parts working correctly. This overhead, plus conflict for shared resources, lowers the expected gain from additional processors. Similarly, a group of N programmers working closely together does not result in N times the amount of work being accomplished.
- **Economy of scale**: Multiprocessor systems can save more money than multiple single-processor systems, because they can share peripherals, mass storage, and power supplies. If several programs operate on the same set of data, it is cheaper to

store those data on one disk and to have all the processors share them, than to have many computers with local disks and many copies of the data.

- **Increased reliability**: If functions can be distributed properly among several processors, then the failure of one processor will not halt the system, only slows down. If we have ten processors and one fails, then each of the remaining nine processors must take a share of the work of the failed processor. Thus, the entire system runs only 10 percent slower, rather than failing altogether. This ability to continue providing service proportional to the level of surviving hardware is called ***graceful degradation***. Systems designed for graceful degradation are also called ***fault tolerant***.

Continued operation in the presence of failures requires a mechanism to allow the failure to be detected, diagnosed, and, if possible, corrected. The Tandem system uses both hardware and software duplication to ensure continued operation despite faults. The system consists of two identical processors, each with its own local memory. The processors are connected by a bus. One processor is the primary and the other is the backup. Two copies are kept of each process: one on the primary processor and the other on the backup. At fixed checkpoints in the execution of the system, the state information of each job including a copy of the memory image-is copied from the primary machine to the backup. If a failure is detected, the backup copy is activated and is restarted from the most recent checkpoint. This solution is expensive, since it involves considerable hardware duplication.

The most common multiple-processor systems now use symmetric multiprocessing (SMP), in which each processor runs an identical copy of the OS, and these copies communicate with one another as needed. Some systems use asymmetric multiprocessing, in which each processor is assigned a specific task. A master processor controls the system; the other processors either look to the master for instruction or have predefined tasks. This scheme defines a master-slave relationship. The master processor schedules and allocates work to the slave processors. SMP means that all processors are peers; no master-slave relationship exists between processors. Each processor concurrently runs a copy of the operating system.

A typical SMP architecture is shown in Figure 1.17. An example of the SMP system is Encore's version of UNIX for the Multimax computer. This computer can be configured such that it employs dozens of processors, all running copies of UNIX. The benefit of this model is that many processes can run simultaneously-N processes can run if there are N CPUs-without causing a significant deterioration of performance. However, we must carefully control I/O to ensure that the data reach the appropriate processor. Also, since the CPUs are separate, one may be sitting idle while another is overloaded, resulting in inefficiencies. These inefficiencies can be avoided if the processors share certain data structures. A multiprocessor system of this form will allow processes and resources to be shared dynamically among the various processors. Almost all modern OS including Windows NT, Solaris, Digital UNIX, OS/2, and Linux will support SMP.

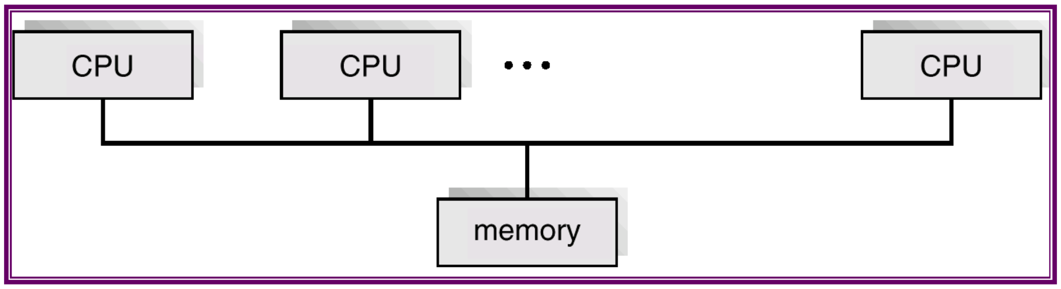


Figure 1.17 Symmetric multiprocessing architecture

The difference between symmetric and asymmetric multiprocessing may be the result of either hardware or software. Special hardware can differentiate the multiple processors, or the software can be written to allow only one master and multiple slaves. For instance, Sun's operating system SunOS Version 4 provides asymmetric multiprocessing, whereas Version 5 (Solaris 2) is symmetric on the same hardware.

As microprocessors become less expensive and more powerful, additional OS functions are off-loaded to slave processors (or back-ends). For example, it is fairly easy to add a microprocessor with its own memory to manage a disk system. The microprocessor could receive a sequence of requests from the main CPU and implement its own disk queue and scheduling algorithm. Thus, the main CPU is relieved from overhead of disk scheduling.

## 1.12 DISTRIBUTED SYSTEMS

Distributed systems depend on networking for their functionality. A network is a communication path between two or more systems. By being able to communicate, distributed systems are able to share computational tasks, and provide a rich set of features to users.

Networks vary by the protocols used, the distances between nodes, and the transport media. TCP/IP is the most common network protocol. Similarly, OS support of protocols varies. Most OS support TCP/IP, including Windows and UNIX. Some systems support proprietary protocols to suit their needs. To an OS, a network protocol needs an interface device (for example, a network adapter) with a device driver to manage it, and software for sending/receiving packet data.

Networks are typecast based on the distances between their nodes; for example, LAN, WAN, MAN etc. The media to carry networks are equally varied. They include copper wires, fiber strands, and wireless transmissions between satellites, microwave dishes and radios. When computing devices are connected to cellular phones, they create a network. Even very short-range infrared communication can be used for networking. These networks also vary by their performance and reliability.

### 1.12.1 Client-Server Systems

As PCs have become faster, more powerful, and cheaper, designers have shifted away from the centralized system architecture. Terminals connected to centralized systems are now being replaced by PCs. Similarly, user-interface functionality that was handled directly by the centralized systems is now being handled by the PCs. As a result, centralized systems today act as server systems to satisfy requests generated by client systems. The general structure of a client-server system is shown in Figure 1.18. Server systems can be broadly categorized as follows:

- **Compute-server systems** provide an interface to which clients can send requests to perform an action, in response to which they execute the action and send back results to the client.
- **File-server systems** provide a file-system interface where clients can create, update, read, and delete files.

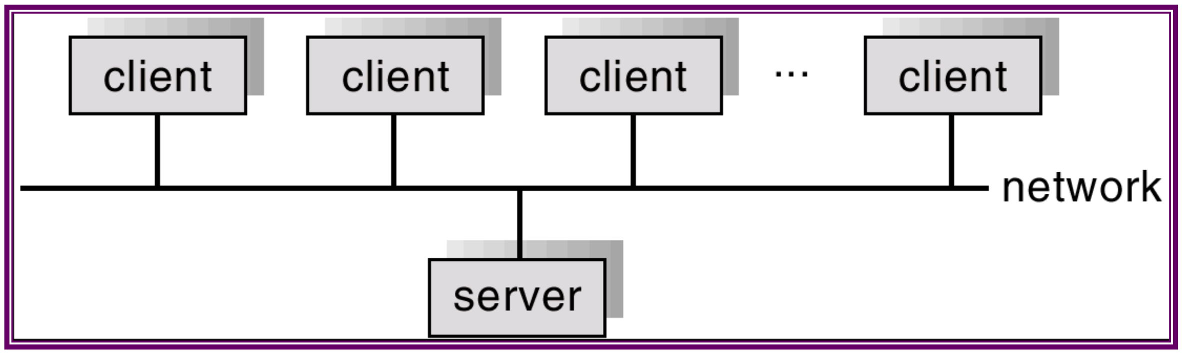


Figure 1.18 General structure of client-server system

### 1.12.2 Peer-to-peer Systems

The growth of computer networks like the Internet and World Wide Web (WWW) has major influence on the recent development of OS. When PCs were introduced in the 1970s, they were designed for personal use. With the beginning of internet for electronic mail, ftp, and gopher, many PCs became connected to computer networks in 1980s. With the introduction of the Web in the mid 1990s, network connectivity became an essential component of a computer system.

Virtually all modern PCs and workstations are capable of running a web browser. OS like Windows, OS/2, MacOS, UNIX etc also include the system software (such as TCP/IP and PPP) that enables a computer to access the Internet via a LAN or telephone connection.

In contrast to the tightly coupled systems discussed earlier, the computer networks used in these applications consist of a collection of processors that do not share memory or a clock. Instead, each processor has its own local memory. The processors communicate with one another through various communication lines, such as high-speed buses or telephone lines. These systems are usually referred to as ***loosely coupled systems*** (or distributed systems).

A network OS is an OS that provides features such as file sharing across the network, and that includes a communication scheme that allows different processes on different computers to exchange messages. A computer running a network OS acts autonomously from all other computers on the network, although it is aware of the network and is able to communicate with other networked computers. A distributed OS is a less autonomous environment. Here, the different OS communicate closely enough to provide the illusion that only a single operating system controls the network.

## 1.13 CLUSTERED SYSTEMS

In clustered systems, multiple CPUs are gathered together to perform computational work. Clustered systems are composed of two or more individual systems coupled together. The clustered computers share storage and are closely linked via LAN networking.

Clustering is usually performed to provide high availability. A layer of cluster software runs on the cluster nodes. Each node can monitor one or more of the others (over the LAN). If the monitored machine fails, the monitoring machine can take ownership of its storage, and restart the application(s) that were running on the failed machine. The failed machine can remain down, but the users and clients of the application would only see a brief interruption of service.

In asymmetric clustering, one machine is in hot standby mode while the other is running the applications. The hot standby host (machine) just monitors the active server. If that server fails, the hot standby host becomes the active server. In symmetric mode, two or more hosts are running applications, and they are monitoring each other. This mode is obviously more efficient, as it uses all of the available hardware.

Other forms of clusters include parallel clusters and clustering over a WAN. Parallel clusters allow multiple hosts to access the same data on the shared storage. Since most OS lack support for this simultaneous data access by multiple hosts, parallel clusters are usually accomplished by special versions of software and special releases of applications. For example, Oracle Parallel Server is a version of Oracle's database that has been designed to run on parallel clusters. Each machine runs Oracle, and a layer of software tracks access to the shared disk. Each machine has full access to all data in the database.

Most systems do not offer general-purpose distributed file systems. Therefore, many clusters do not allow shared access to data on the disk. For this, distributed file systems must provide access control and locking to the files to ensure no conflicting operations occur. This type of service is commonly known as a ***distributed lock manager (DLM).***

Cluster technology is rapidly changing. Cluster directions include global clusters, in which the machines could be anywhere in the. Such projects are still the subject of research and development. Clustered system use and features should expand greatly as storage-area networks (SANS) become prevalent. SANs allow easy attachment of multiple hosts to multiple storage units. Current clusters are usually limited to two or four hosts due to the complexity of connecting the hosts to shared storage.

## 1.14 REAL-TIME SYSTEMS

A real-time system is used when there is a time requirements on the operation of a processor or the flow of data. Thus, it is used as a control device in a dedicated application. Sensors bring data to the computer. The computer must analyze the data and possibly adjust controls to modify the sensor inputs. Systems that control scientific experiments, medical imaging systems, industrial control systems, and certain display systems are real-time systems. Some automobile-engine fuel-injection systems, home-appliance controllers, and weapon systems are also real-time systems.

A real-time system has well-defined, fixed time constraints. Processing must be done within the defined constraints, or the system will fail. For instance, a robot arm must be instructed to halt before it reaches a wall. A real-time system functions correctly only if it returns the correct result within its time constraints. So in real-time system, a quick response is expected; whereas in batch system time constraints are not at all there.

Real-time systems may be hard or soft. A hard real-time system guarantees that critical tasks be completed on time. That is, various tasks like retrieval of stored data, process by OS etc are bounded by time. Secondary storage is usually limited with data being stored in read-only memory (ROM). ROM is located on nonvolatile storage devices that retain their contents even in the case of electric outage; most other types of memory are volatile. Most advanced OS features are absent too, since they tend to separate the user from the hardware. And this result in uncertainty about the amount of time an operation will take. For example, virtual memory is almost never found on real-time systems. Therefore, hard real-time systems conflict with the operation of time-sharing systems, and the two cannot be mixed. None of the existing general-purpose OS support hard real-time functionality.

A soft real-time system is less restrictive, where a critical real-time task gets priority over other tasks, and retains that priority until it completes. Here also, the OS kernel delays need to be bounded: A real-time task cannot be kept waiting indefinitely for the kernel to run it. Soft real time is an achievable goal that can be mixed with other types of systems. But, soft real-time systems have limited utility than hard real-time systems. Given their lack of deadline support, they are risky to use for industrial control and robotics. They are useful in areas like multimedia, virtual reality, and advanced scientific projects-such as undersea exploration and planetary rovers. These systems need advanced OS features that cannot be supported by hard real-time systems. Because of the expanded uses for soft real-time functionality, it is finding its way into most current operating systems, including major versions of UNIX.

## 1.15 HANDHELD SYSTEMS

Handheld systems include personal digital assistants (PDAs), cell phones, tablets with/without internet connection. Developers of handheld systems and applications face many challenges, most of which are due to the limited size of such devices. Due to limited size, most handheld devices have a small amount of memory, slow processors, and small display screens. The handheld devices have following limitations:

- **Limited Memory:** Many handheld devices have between 512 KB and 2 GB of memory. As a result, the operating system and applications must manage memory efficiently. This includes returning all allocated memory back to the memory manager once the memory is no longer being used. Virtual memory allows developers to write programs that behave as if the system has more memory than actual availability. Currently, many handheld devices do not use virtual memory techniques, thus forcing program developers to work within the limited physical memory.

- **Less Speed:** Processors for most handheld devices run at a fraction of the speed of a processor in a PC. Faster processors require more power. To include a faster processor in a handheld device would require a larger battery that would have to be replaced (or recharged) more frequently. To minimize the size of most handheld devices, smaller, slower processors which consume less power are typically used. Therefore, the OS and applications must be designed accordingly.
- **Small Display:** Handheld devices have a display of about 3-5 inches, whereas, PCs will be having display of about 14 – 40 inches. Familiar tasks, such as reading e-mail or browsing web pages, must be condensed onto smaller displays. One approach for displaying the content in web pages is web clipping, where only a small subset of a web page is delivered and displayed on the handheld device.

## 1.16 FEATURE MIGRATION

Overall examination of OS for mainframes and microcomputers shows that features once available only on mainframes have been adopted by microcomputers. The same concepts are appropriate for the various classes of computers: mainframes, minicomputers, microcomputers, and handhelds. Figure 1.19 shows the migration of OS features. However, to start understanding modern OS, you need to realize the theme of feature migration and to recognize the long history of many OS features.

A good example of this movement occurred with the MULTIplexed Information and Computing Services (MULTICS) operating system. MULTICS was developed from 1965 to 1970 at the Massachusetts Institute of Technology (MIT) as a computing utility. It ran on a large, complex mainframe computer (the GE 645). Many of the ideas that were developed for MULTICS were subsequently used at Bell Laboratories (one of the original partners in the development of MULTICS) in the design of UNIX. The UNIX operating system was designed circa 1970 for a PDP-11 minicomputer. Around 1980, the features of UNIX became the basis for UNIX-like operating systems on microcomputer systems, and they are being included in more recent operating systems such as Microsoft Windows NT, IBM 0S/2, and the Macintosh operating system. Thus, the features developed for a large mainframe system have moved to microcomputers over time.

At the same time as features of large operating systems were being scaled down to fit PCs, more powerful, faster, and more sophisticated hardware systems were being developed. The personal workstation is a large PC-for example, the Sun SPARCstation, the HP/Apollo, the IBM RS/6000, and the Intel Pentium class system running Windows NT or a UNIX derivative. Many universities and businesses have large numbers of workstations tied together with local-area networks. As PCs gain more sophisticated hardware and software, the line dividing the two categories-mainframes and microcomputers-is blurring.

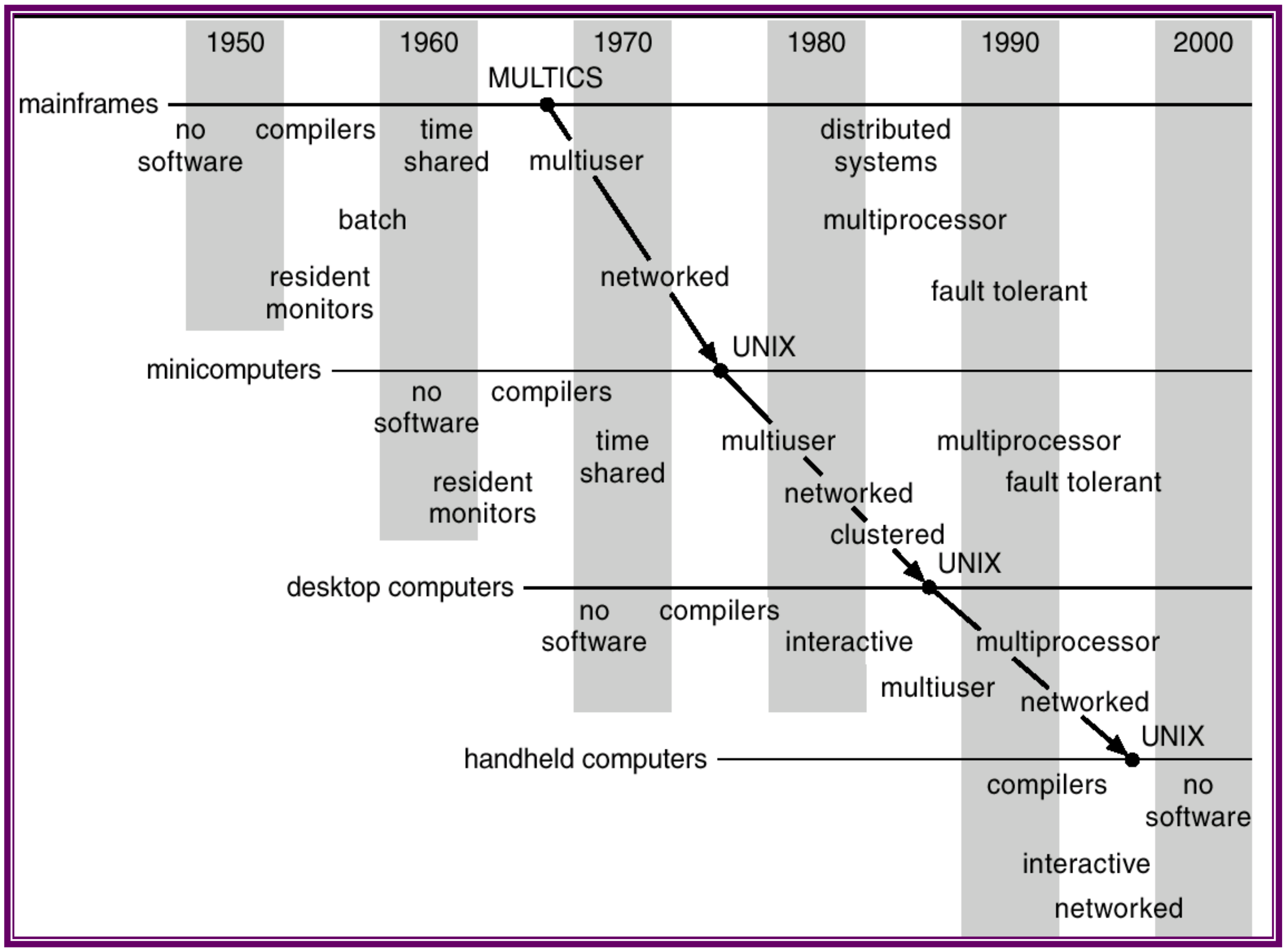


Figure 1.19 Migration of OS concepts and features

## 1.17 COMPUTING ENVIRONMENTS

Here, we will discuss how multiprogrammed, time-shared, handheld computers are used in variety of computing environment settings.

### 1.17.1 Traditional Computing

As the time passes by, the computing environment is getting matured. Just a few years ago, a typical office environment consisted of PCs connected to a network, with servers providing file and print service. Remote access was not so easy, and portability was achieved by carrying laptops. Terminals attached to mainframes were common at many companies as well, with even fewer remote access and portability options.

The current trend is toward more ways to access these environments. Web technologies are stretching the boundaries of traditional computing. Companies implement portals which provide web accessibility to their internal servers. Network computers are essentially terminals that understand web-based computing. Handheld computers can synchronize with PCs to allow very portable use of company information. They can also connect to wireless networks to use the company's web portal.

At home, most users had a single computer with a slow modem connection to the office and Internet. Network connections with good speed are now available for lower cost. Those fast data connections are allowing home computers to serve up web pages and to contain their own networks with printers, client PCs, and servers. Some homes even have firewalls to protect these home environments from security breaches.

### 1.17.2 Web-based Computing

The Web has become ubiquitous, leading to more access by a wider variety of devices than was imagined few years ago. PCs are still the most common access devices, with workstations (high-end graphics-oriented PCs), handheld PDAs, and even cell phones also providing access.

Web computing has increased the emphasis on networking. Now, devices have wired/wireless networks with faster connectivity. The implementation of web-based computing has given rise to new categories of devices, such as load balancers which distribute network connections among a pool of similar servers. Operating systems like Windows 95, which acted as web clients, have evolved into Windows ME and Windows 2000, which can act as web servers as well as clients. Generally, the Web has increased the complexity of devices as their users require them to be web-enabled.

### 1.17.3 Embedded Computing

Embedded computers are the most common form of computers in existence. They run embedded real-time OS. These devices are found everywhere, from car engines and manufacturing robots to VCRs and microwave ovens. They tend to have very specific tasks. The systems they run on are usually primitive, lacking advanced features, such as virtual memory, and even disks. Thus, the OS provide limited features. They usually have little or no user interface, preferring to spend their time monitoring and managing hardware devices, such as automobile engines and robotic arms.

As an example, consider firewalls and load balancers. Some are general-purpose computers, running standard OS like UNIX-with special-purpose applications loaded to implement the functionality. Others are hardware devices with a special-purpose OS embedded within, providing just the functionality desired.

The use of embedded systems will increase over the time. Their need as a standalone device or as a member of network/web is increasing. Entire houses can be computerized, so that a central computer-either a general-purpose computer or an embedded system-can control heating and lighting, alarm systems, and even coffee makers. Web access can let a home-owner tell the house to heat up before he arrives home. Someday, the refrigerator may call the grocery store when it notices the milk is gone.