Uttar Pradesh Rajarshi Tandon Open University

MBA -3.51

Computer Fundamentals and its Organisation

BLOCK

1

Computer Basics & Data Representation

Unit-I **Computer Basics**	**5**
Unit-II **Problem Solving**	**21**
Unit-III **Data Representation**	**26**
Unit-IV **Hexadecimal Representation**	**39**

Printed and Published by **Prof. P.P. Dubey**, Registrar, Uttar Pradesh Rajarshi Tandon Open University , Prayagraj, 2022.

Printed By : K.C.Printing & Allied Works, Panchwati, Mathura -281003.

Block - 1 : Computer Basics & Data Representation

This is the first block of your course. It deals with the computer basics & data representation. This block provide you the basics of computer and also tell you how data represent in to the computer. Data representation plays a very important role in the field of the computer. For this reason we divided this block into four following units.

So we will start the first unit with basics of algorithms. We also focused how to developed an algorithm, how to calculate its complexity and its characteristics. We also discussed a simple model of a computer. We describe the components of computer system. We also deals with the memories of computer. In this unit we also described the characteristics of computers.

In the second unit we focused on problem-solving using computers. We describe this problem solving with the help of algorithm, flowcharting, and pseudo code.

In the third unit we told about the data representation in computer. We also describe how characters representation done in the computer system. As we all know the role of the numbers in computer system so we focused on the representation of integers and fractions.

In the fourth unit we focused on hexadecimal representation of numbers. We also focused on the conversion between octal and hexadecimal, decimal and binary, and discussed error-detecting codes.

As you study the material you will come across abbreviations in the text, e.g. Sec. 1.1, Eq.(1 .I) etc. The abbreviation Sec. stands for section, and Eq. for equation. Figure, a. b refers to the bth figure of Unit a, i.e. Figure. 1.1 is the first figure in Unit 1. Similarly, Sec. 1.1 is the first section in Unit 1 and Eq.($.8) is the eighth equation in Unit 4. Similarly Table x. y refers to the yth table of Unit x, i.e. Table. 1.1 is the first table in Unit 1.

In your study you will also find that the units are not of equal-length and your study time for each unit will vary.

We hope you enjoy studying the material and wish you success.

UNIT-I

Computer Basics

Structure

1.0 Introduction

1.1 Algorithms

1.2 A Simple Model of a Computer5

1.3 Characteristics of Computers

1.4 Summary

1.5 Review Questions

1.0 Introduction

In this unit we define about the basics of computer. In this unit there are five sections. In section one i.e. Sec. 1.1 you will learn about algorithm. As you have studied earlier that an algorithm (pronounced AL-go-rith-um) is a procedure or formula for solving a problem. The word derives from the name of the mathematician, Mohammed ibn-Musa al-Khwarizmi, who was part of the royal court in Baghdad and who lived from about 780 to 850. Al-Khwarizmi's work is the likely source for the word algebra as well.

A computer program can be viewed as an elaborate algorithm. In mathematics and computer science, an algorithm usually means a small procedure that solves a recurrent problem. In Sec. 1.2 you will know about the basics of computer. In this section we define input unit, output unit, CPU, ALU, CU etc. in the next section i.e. Sec. 1.3 we explain characteristics of computer. In Sec. 1.4 and 1.5 you will find summary and review questions respectively.

Objectives

After studying this unit you should be able to:

- Define algorithm, analysis of algorithm, characteristics of algorithm
- Express basics of computer like CPU, CU, ALU, input and output unit.
- Describe characteristics of computer.

1.1 Algorithm

An algorithm, named after the ninth century scholar *Abu Jafar Muhammad Ibn Musu Al-Khowarizmi*, is defined as follows: Roughly speaking:

- An algorithm is a set of rules for carrying out calculation either by hand or on a machine.
- An algorithm is a finite step-by-step procedure to achieve a required result.
- An algorithm is a sequence of computational steps that transform the input into the output.
- An algorithm is a sequence of operations performed on data that have to be organized in data structures.
- An algorithm is an abstraction of a program to be executed on a physical machine (model of Computation).

The most famous algorithm in history dates well before the time of the ancient Greeks: this is the Euclid's algorithm for calculating the greatest common divisor of two integers. This theorem appeared as the solution to the Proposition II in the Book VII of Euclid's "Elements." Euclid's "Elements" consists of thirteen books, which contain a total number of 465 propositions.

Input
Algorithm
Output

Figure 1-1

"A step-by-step problem-solving procedure, especially an established, recursive computational procedure for solving a problem in a finite number of steps"

The Classic Multiplication Algorithm

1. Multiplication, the American way:

Multiply the multiplicand one after another by each digit of the multiplier taken from right to left.

```
          9 8 1
        1 2 3 4
-----------------------
        3 9 2 4
      2 9 4 3
    1 9 6 2
    9 8 1
-----------------------
  1 2 1 0 5 5 4
```

Figure 1-2

2. Multiplication, the English way:

Multiply the multiplicand one after another by each digit of the multiplier taken from left to right.

```
          9 8 1
        1 2 3 4
-----------------------
    9 8 1
    1 9 6 2
      2 9 4 3
        3 9 2 4
-----------------------
  1 2 1 0 5 5 4
```

Figure 1-3

Algorithmic is a branch of computer science that consists of designing and analyzing computer algorithms

1. The "design" pertains to

    i. The description of algorithm at an abstract level by means of a pseudo language, and
    ii. Proof of correctness that is, the algorithm solves the given problem in all cases.

2. The "analysis" deals with performance evaluation (complexity analysis).

We start with defining the model of computation, which is usually the Random Access Machine (RAM) model, but other models of computations can be used such as PRAM. Once the model of computation has been defined, an algorithm can be describe using a simple language (or pseudo language) whose syntax is close to programming language such as C or java.

Algorithm's Performance

Two important ways to characterize the effectiveness of an algorithm are its space complexity and time complexity. Time complexity of an algorithm concerns determining an expression of the number of steps needed as a function of the problem size. Since the step count measure is somewhat coarse, one does not aim at obtaining an exact step count. Instead, one attempts only to get asymptotic bounds on the step count. Asymptotic analysis makes use of the O (Big Oh) notation. Two other notational constructs used by computer scientists in the analysis of algorithms are Θ (Big Theta) notation and Ω (Big Omega) notation.

The performance evaluation of an algorithm is obtained by totaling the number of occurrences of each operation when running the algorithm. The performance of an algorithm is evaluated as a function of the input size n and is to be considered modulo a multiplicative constant.

The following notations are commonly use notations in performance analysis and used to characterize the complexity of an algorithm.

Θ-Notation (Same order)

This notation bounds a function to within constant factors. We say $f(n) = \Theta(g(n))$ if there exist positive constants n_0, c_1 and c_2 such that to the right of n_0 the value of $f(n)$ always lies between $c_1\ g(n)$ and $c_2\ g(n)$ inclusive.

In the set notation, we write as follows:

$\Theta(g(n)) = \{f(n)$: there exist positive constants c_1, c_1, and n_0 such that $0 \le c_1\ g(n) \le f(n) \le c_2\ g(n)$ for all $n \ge n_0\}$

We say that is $g(n)$ an asymptotically tight bound for $f(n)$.

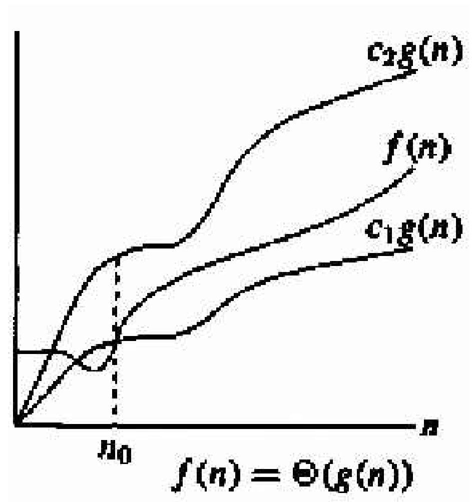


Figure 1-4

Graphically, for all values of n to the right of n_0, the value of $f(n)$ lies at or above $c_1\ g(n)$ and at or below $c_2\ g(n)$. In other words, for all $n \ge n_0$, the function $f(n)$ is equal to $g(n)$ to within a constant factor. We say that $g(n)$ is an asymptotically tight bound for $f(n)$.

In the set terminology, $f(n)$ is said to be a member of the set $\Theta(g(n))$ of functions. In other words, because $O(g(n))$ is a set, we could write

$f(n) \in \Theta(g(n))$

to indicate that $f(n)$ is a member of $\Theta(g(n))$. Instead, we write

$f(n) = \Theta(g(n))$

to express the same notation.

Historically, this notation is "$f(n) = \Theta(g(n))$" although the idea that $f(n)$ is equal to something called $\Theta(g(n))$ is misleading.

Example: $n^2/2 - 2n = (n^2)$, with $c_1 = 1/4$, $c_2 = 1/2$, and $n_0 = 8$.

O-Notation (Upper Bound)

This notation gives an upper bound for a function to within a constant factor. We write $f(n) = O(g(n))$ if there are positive constants n_0 and c such that to the right of n_0, the value of $f(n)$ always lies on or below $c\,g(n)$.

In the set notation, we write as follows: For a given function $g(n)$, the set of functions

$O(g(n)) = \{f(n)$: there exist positive constants c and n_0 such that $0 \le f(n) \le c\,g(n)$ for all $n \ge n_0\}$

We say that the function $g(n)$ is an asymptotic upper bound for the function $f(n)$. We use O-notation to give an upper bound on a function, to within a constant factor.

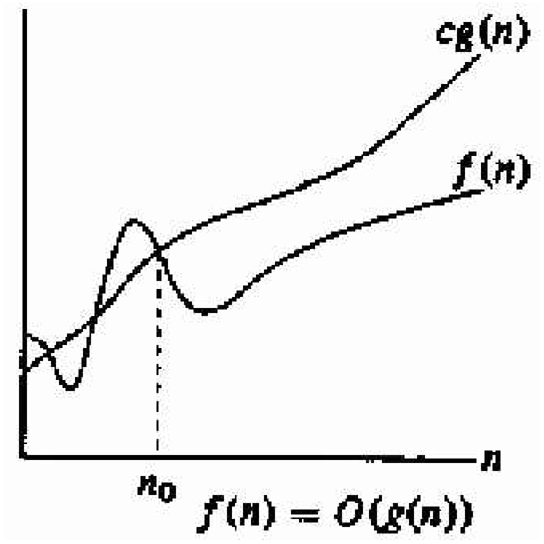


Figure 1-5

Graphically, for all values of n to the right of n_0, the value of the function $f(n)$ is on or below g(n). We write $f(n) = O(g(n))$ to indicate that a function $f(n)$ is a member of the set $O(g(n))$ i.e.

$f(n) \in O(g(n))$

Note that $f(n) = \Theta(g(n))$ implies $f(n) = O(g(n))$, since Θ-notation is a stronger notation than O-notation.

Example: $2n^2 = O(n^3)$, with $c = 1$ and $n_0 = 2$.

Equivalently, we may also define f is of order g as follows:

If $f(n)$ and $g(n)$ are functions defined on the positive integers, then $f(n)$ is $O(g(n))$ if and only if there is a $c > 0$ and an $n_0 > 0$ such that

$|f(n)| \le |g(n)|$ for all $n \ge n_0$

Historical Note: The notation was introduced in 1892 by the German mathematician Paul Bachman.

Ω-Notation (Lower Bound)

This notation gives a lower bound for a function to within a constant factor. We write $f(n) = \Omega(g(n))$ if there are positive constants n_0 and c such that to the right of n_0, the value of $f(n)$ always lies on or above $c\,g(n)$.

In the set notation, we write as follows: For a given function $g(n)$, the set of functions

$\Omega(g(n)) = \{f(n)$: there exist positive constants c and n_0 such that $0 \leq c\,g(n) \leq f(n)$ for all $n \geq n_0\}$

We say that the function $g(n)$ is an asymptotic lower bound for the function $f(n)$.

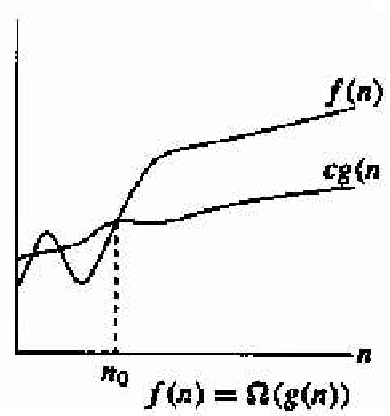


Figure 1-6

The intuition behind Ω-notation is shown above.

Example: $\sqrt{n} = (\lg n)$, with $c = 1$ and $n_0 = 16$.

Algorithm Analysis

The complexity of an algorithm is a function $g(n)$ that gives the upper bound of the number of operation (or running time) performed by an algorithm when the input size is n.

There are two interpretations of upper bound.

Worst-case Complexity

The running time for any given size input will be lower than the upper bound except possibly for some values of the input where the maximum is reached.

Average-case Complexity

The running time for any given size input will be the average number of operations over all problem instances for a given size.

Because, it is quite difficult to estimate the statistical behavior of the input, most of the time we content ourselves to a worst case behavior. Most of the time, the complexity of $g(n)$ is approximated by its family $o(f(n))$ where $f(n)$ is one of the following functions. n (linear complexity), $\log n$ (logarithmic complexity), n^a where $a \geq 2$ (polynomial complexity), a^n(exponential complexity).

Optimality

Once the complexity of an algorithm has been estimated, the question arises whether this algorithm is optimal. An algorithm for a given problem is optimal if its complexity reaches the lower bound over all the algorithms solving this problem. For example, any algorithm solving "the intersection of n segments" problem will execute at least n^2 operations in the worst case even if it does nothing but print the output. This is abbreviated by saying that the problem has $\Omega(n^2)$ complexity. If one finds an $O(n^2)$ algorithm that solve this problem, it will be optimal and of complexity $\Theta(n^2)$.

Reduction

Another technique for estimating the complexity of a problem is the transformation of problems, also called problem reduction. As an example, suppose we know a lower bound for a problem A, and that we would like to estimate a lower bound for a problem B. If we can transform A into B by a transformation step whose cost is less than that for solving A, then B has the same bound as A.

The Convex hull problem nicely illustrates "reduction" technique. A lower bound of Convex-hull problem established by reducing the sorting problem (complexity: $\Theta(n \log n)$) to the Convex hull problem.

Methodologies for Analyzing Algorithms

We will be primarily concerned with the speed (*time complexity*) of algorithms.

- Sometimes the *space complexity* is studied.
- The time depends on the input, most often on the size of the input.
- We can run experiments.
    - Must choose *sufficiently many, representative* inputs.
    - Must use identical hardware to compare algorithms.
    - Must *implement* the algorithm.

We will emphasize instead and analytic framework that is independent of input and hardware, and does not require an implementation. The disadvantage is that we can only estimate the time required.

- Often we ignore multiplicative constants and small input values.
- So we consider $f(x)=x^3-20x^2$ equivalent to $g(x)=10x^3+10x^2$
- Huh??
- Easy to see that for say $x > 100$, $f(x) < 10\ g(x)$ and $g(x) < 10\ f(x)$.

Method for Developing an Algorithm

1. Define the problem: State the problem you are trying to solve in clear and concise terms.

2. List the inputs (information needed to solve the problem) and the outputs (what the algorithm will produce as a result)

3. Describe the steps needed to convert or manipulate the inputs to produce the outputs.

Start at a high level first, and keep refining the steps until they are effectively computable operations.

4. Test the algorithm: choose data sets and verify that your algorithm works!

Characteristics of an Algorithm

What makes an algorithm an algorithm? There are four essential properties of an algorithm.

1. Each step of an algorithm must be exact.

   This goes without saying. An algorithm must be precisely and unambiguously described, so that there remains no uncertainty. An instruction that says "shuffle the deck of card" may make sense to some of us, but the machine will not have a clue on how to execute it, unless the detail steps are described. An instruction that says "lift the restriction" will cause much puzzlement even to the human readers.

2. An algorithm must **terminate.**

   The ultimate purpose of an algorithm is to solve a problem. If the program does not stop when executed, we will not be able to get any result from it. Therefore, an algorithm must contain a finite number of steps in its execution. Note that an algorithm that merely contains a finite number of steps may not terminate during execution, due to the presence of 'infinite loop'.

3. An algorithm must be **effective.**

   Again, this goes without saying. An algorithm must provide the correct answer to the problem.

4. An algorithm must be **general.**

   This means that it must solve every instance of the problem. For example, a program that computes the area of a rectangle should work on all possible dimensions of the rectangle, within the limits of the programming language and the machine.

An algorithm should also emphasize on what's, and not the *how's*, leaving the details for the program version. However, this point is more apparent in more complicated algorithms at advanced level, which we are unlikely to encounter yet.

1.2 A simple Computer Model

A computer is a programmable machine designed to perform arithmetic and logical operations automatically and sequentially on the input given by the user and gives the desired output after processing. Computer components are divided into two major categories namely hardware and software. Hardware is the machine itself and its connected devices such as monitor, keyboard, mouse etc. Software is the set of programs that make use of hardware for performing various functions.

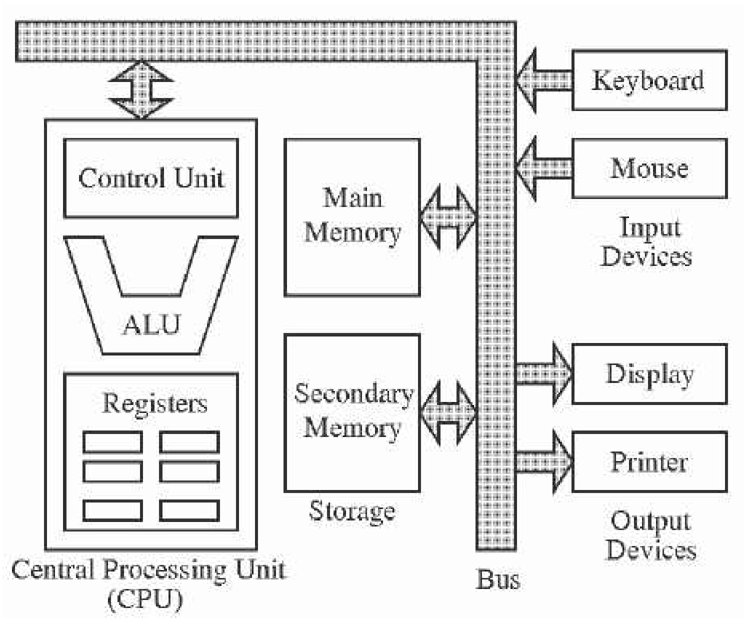


Figure 1.7

There are three major components of a computer:

- Input/Output Unit
- Central Processing Unit
    - Control Unit (CU)
    - Arithmetic Logic Unit (ALU)
    - Registers
- Memory
- Output Unit

INPUT UNIT

This unit contains devices with the help of which we enter data into computer. This unit is linked between user and computer. Input devices translate the human-being information into the form understandable by a computer.

The input, output and storage devices are described as on line, when they are directly connected to the CPU, when not connected directly they are described as off-line.

Examples of input devices:

- Keyboard
- Microphone
- Mouse
- light pen
- Scanner
- Joystick
- Voice recognizer
- Card reader
- Digitizer
- Floppy drive
- Tape drive
- Cartridge tape drives
- OCR (Optical Character Reader)
- OMR (Optical Mark Reader) etc.

OUTPUT UNIT

Output unit consists of devices with the help of which we get the information from computer. Output unit is a link between computer and user. Output devices translate the computer's output into the form understandable by user.

Examples of Output devices:

- VDU
- Line Printer
- Dot matrix printer
- Daisy wheel printer
- Laser printer
- Color graphic terminal
- Graph plotter
- Floppy drive
- Tape drive
- Disk drive
- Cartridge tape drive etc.

Central Processing Unit (CPU)

The central processing unit (CPU) is the brain of any computer. It carries out all the processing in the computer. Central Processing Unit itself consists of three main subsystems. *The first one is Control Unit, the second is Arithmetic and Logic Unit (ALU), and the third is Registers.*

A CPU works in a fetch execute cycle. On power on, the CPU fetches the first instruction from a location specified by the program counter. This instruction is brought into instruction register which is decoded by the control unit. Based on the instruction, the control unit would fetch the operand and or carry out arithmetic or logical operations on it, or store the result of such an operation into a specified memory location. After one instruction is executed the next instruction is fetched by the processor and executed. This process goes on till the processor does not come to a halt instruction. A real life processor would have large number of registers, sophisticated micro-program control unit and a sophisticated arithmetic and logic unit. Most powerful processors currently popular are from Intel, Pentium III and Pentium IV, Dual Core, Core2Duo, Core i3, Core i5 and Core i7.

Figure 1-8

CONTROL UNIT

Control unit controls the operations of all parts of computers. It does not carry out any actual data processing operations.

- It retrieves instructions from the main memory and determines what is to be taken.
- It then retrieves the data required to be processed from the main memory.
- It causes the CPU to actually carry out required operations and determine whether the required operation have been carried out or not.
- It places the processed results in the output area of the memory.
- It fetches the next instruction from the memory and repeat the whole cycle of operations outlined above.

In addition to the above, the control unit also oversees that erroneous data does not enter into the system (for example, numeric data consisting of alphabets or a number is divided by zero). When such an event occurs, the control unit displays an error on the screen of the CPU to warn computer operator.

In order to carry out these operations, the control unit also has its own set of registers (like those of

ALU). The basic register of the control unit are the instruction register, the decoder & the address register.

ALU (Arithmetic and Logic Unit)

This unit consists of two subsections:

- Arithmetic section
- Logic section

Arithmetic Section: Function of Arithmetic section is to perform arithmetic operations like addition, subtraction, multiplication & division. All complex operation are done by making repetitive use of above operation.

Logic Section: Function of logic section is to perform logic operations such as comparing, selecting, matching and merging of data.

The arithmetic and logic unit (ALU) contains a number of storage locations referred to as registers. These registers are composed of electronic circuitry having the capability of adding, subtracting, multiplying rounding off etc., the number of registers in a computer vary from model to model. However, the basic registers in any computer are the adder and the accumulator.

An interesting side limit is that a computer can only add. It cannot carry out subtraction, Multiplication and division operating in the way it is done manually. For these operations, it also has to take the add route. Thus if 15 to be multiplied by 10, the computer adds the data item 15 times, For subtraction and division, it employs the 1's complement method, which again is a form of the addition process. The basic add & subtract operation have been carried out by the computer by means of dedicated circuits called micro-programs.

Registers

The Registers within CPU are special purpose temporary storage locations. The important registers, within CPU are, program counter (PC). A program counter keeps track of next instruction to be executed. Instruction Register (IR) is a register which holds instruction to be decoded by the control unit. Then memory address register (MAR), is a register which points to the memory location which the CPU plans to access, either for reading or for writing. MBR (memory buffer register) which is also referred to as memory data register (MDR) is used for storage data either coming to the CPU or data being transferred by the CPU. Accumulator (ACC) is a general purpose register used for storing operands, temporary results and results produced by arithmetic logic unit. Besides these a processor can have many other registers. But these are the most basic and most essential registers necessary for any CPU.

MEMORY OR STORAGE UNIT

The function of storage unit is to store instruction, data and intermediate results. This unit supplies information to the other units of the computer when needed. It is also known as *internal storage unit* or *main memory* or *primary storage*. Memory is part of the main computer system. The processor access the main memory in direct fashion, that is, the processor can access any location of this memory either to read information from it or store information in it. The primary memory is implemented by two types of memory technologies. The first is called *random access memory* (RAM) another is *read only memory* (ROM). Its size affects speed, power and capabilities.

Random Access Memory

RAM directly provides the required information to the processor. It can be defined as a block of sequential memory locations, each of which has a unique address determining the location and those locations contain a data element. Storage locations in main memory are addressed directly by the CPU's instructions. It is volatile in nature, as soon as powered turned off, the information stored in it will lost. RAM can be further divided into two categories:

- Dynamic Random Access Memory
- Static Random Access Memory

Dynamic Random Access Memory (DRAM):

This type of memory holds the data in dynamic manner with the help of a refresh circuitry. Each second or even less that contents of each memory cell is read and the reading action refreshing the contents of the memory? Due to refreshing action, this memory is called dynamic RAM.

Static Random Access Memory (SRAM):

SRAM along with DRAM is essential for a system to run optimally, because it is very fast as compared to DRAM. It is effective because most programs access the same data repeatedly and keeping all this information in the first written to SRAM assuming that it will be used again soon. SRAM is generally included in computer system by the name of cache.

Read Only Memory (ROM)

As the name suggests, read only memory can only be read, not written. CPU can only read from any location in the ROM but cannot write. The contents of ROM are not lost even in case of a sudden power failure, making it non-volatile in nature. The instructions in ROM are built into the electronic circuit of the chip. These instructions are called firmware. Read only memory is also random access in nature, which means that CPU can randomly access any location within ROM. Improvement in technology for construction flexible ROM has brought, PROM(Programmable Read Only Memory), EPROM(Erasable Programmable Read Only Memory), and EEPROM(Electrical Erasable Read Only Memory) into existence.

Cache Memory

Cache is a piece of very fast memory, made from high-speed static RAM that reduces the access time of the data. It is very expensive generally incorporated in the processor, where valuable data and program segments are kept. Main reason for introducing cache in between main memory and processor is to compensate the speed mismatch. Figure shows 1.3 the role of cache in memory-processor communication

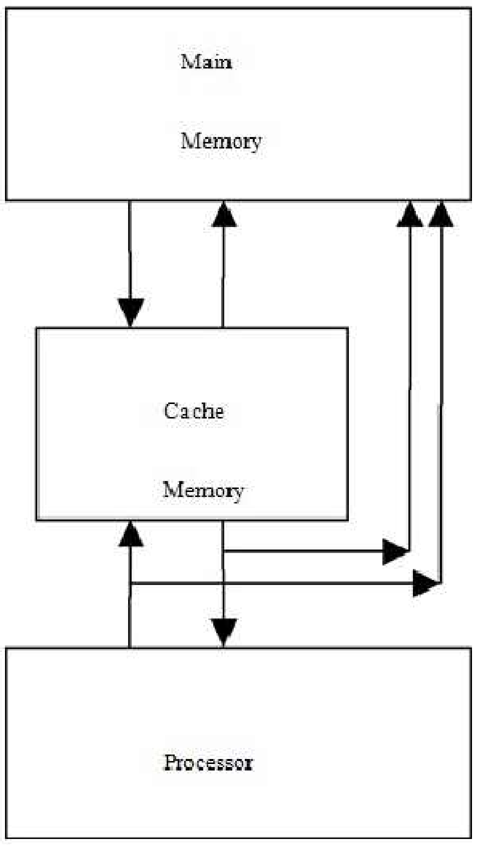


Figure 1.9

Secondary Memory

Secondary storage provides large, non-volatile, and inexpensive storage for programs and data. However, the access time in secondary memory is much larger than in primary memory. Secondary storage permits the storage of computer instructions and data for long periods of time. Secondary storage is also called auxiliary or bulk memory. Magnetic disks (Hard disks, floppy disks, CD-RW) and magnetic tape are examples of secondary storage.

Hierarchy of memories

Internal Processor Memories

These consist of set of high-speed registers that are internal to a processor and are used as temporary storage locations to hold data during processing.

Primary Memory or Main Memory

This memory is large as compared to inter processor memory but not as fast. This memory has direct link with internal processor memory.

Secondary Memory or Auxiliary Memory

This memory is much large in size compared to main memory but is slower. There is another kind of memory used in modern computers. It is called **cache** memory. Though it is a part of main memory, it is logically positioned between the internal memory, registers, and main memory. Figure 1.4 shows the hierarchy of memories.

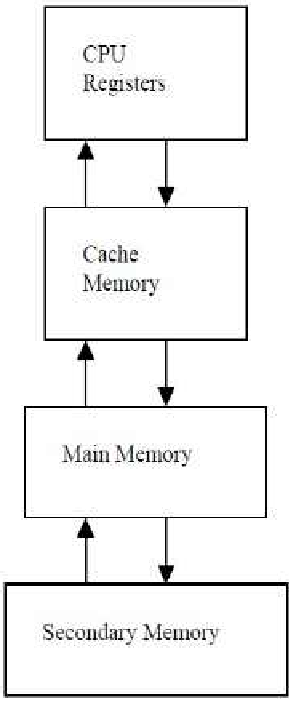


Figure 1.10

Unit of Memory

The various units used to measure computer memory, are as follows:

Bit: Bit, Abbreviation for binary digit, is basic unit of memory. It is smallest unit of information. Bit is represented by a lower case **b.**

Byte: A unit of 8 bits is known as a byte. Hence, a byte is able to contain any binary number between 00000000 and 11111111. It is represented by uppercase **B.**

Kilobyte: One **KB** is equal to 1024 bytes.

Megabyte: One **MB** is equal to 1024 KB.

Gigabyte: One **GB** is equal to 1024 MB.

Terabyte: One **TB** is equal to 1024 GB.

The arrows shows in the figure indicates data flow. In the computer the Buses is used for data flow.

Buses - Essentially wires that connect CPU to memory and Input/Output devices.

Data bus - Carries data between CPU, memory and I/O. Note that data is anything stored or retrieved to or from memory or I/O devices.

Address bus - Carries the address of the memory location or I/O device the CPU is accessing. In this simple model, only the CPU can generate an address.

Control bus - Carries control signals such as read or write memory, an interrupt has occurred, clock signals, etc.

Check your progress

Q1. Define algorithm. How we analyze an algorithm?

Q2. Draw basic diagram of computer system.

Q3. Define all the components of computer system.

1.3 Characteristics of Computers

Increasing popularity of computers has proved that it is a very powerful and useful tool. The power and usefulness of this popular tool are mainly due to its following characteristics:

1. **Automatic-** An automatic machine works by itself without human intervention. Computers are automatic machines because once started on a job, they carry out the job (normally without any human assistance) until it is finished. However, computers being machines cannot start themselves and cannot go out and find their own problems and solutions. We need to instruct a computer using coded instructions that specify exactly how it will do a particular job. Some of the other characteristics of computers (such as speed and accuracy) are because they are automatic and work on a problem without any human intervention.

2. **Speed-** A computer is a very fast device. It can perform in a few seconds, the amount of work that a human being can do in an entire year — if he/she worked day and night and did nothing else. In other words, a computer can do in a few minutes what would take a man his entire lifetime. While talking about the speed of a computer we do not talk in terms of seconds or even milliseconds (10^{-3}) but in terms of microseconds (10^{-6}), nanoseconds (10^{-9}), and even picoseconds (10^{-12}). A powerful computer is capable of performing several billion (10^{9}), even trillion (10^{12}), simple arithmetic operations per second.

3. **Accuracy-** In addition to being very fast, computers are very accurate. Accuracy of a computer is consistently high and the degree of its accuracy depends upon its design. A computer performs every calculation with the same accuracy. However, errors can occur in a computer. These errors are mainly due to human rather than technological weaknesses. For example, errors may occur due to imprecise thinking by a programmer (a person who writes instructions for a computer to solve a particular problem) or incorrect input data. We often refer to computer errors caused due to incorrect input data or unreliable programs as garbage-in-garbage-out (GIGO).

4. **Diligence-** Unlike human beings, a computer is free from monotony, tiredness, and lack of concentration. It can continuously work for hours without creating any error and without grumbling. Hence, computers score over human beings in doing routine type of jobs that require great accuracy. If ten million calculations have to be performed, a computer will perform the last one with exactly the same accuracy and speed as the first one.

5. **Versatility-** Versatility is one of the most wonderful things about a computer. One moment it is preparing results of an examination, next moment it is busy preparing electricity bills, and in between, it may be helping an office secretary to trace an important letter in seconds. All that is required to change its talent is to slip in a new program (a sequence of instructions for the computer) into it. In brief, a computer is capable of performing almost any task, if the task can be reduced to a finite series of logical steps.

6. **Power of remembering-** As a human being acquires new knowledge, his/her brain subconsciously selects what it feels to be important and worth retaining in memory. The brain relegates unimportant details to back of mind or just forgets them. This is not the case with

computers. A computer can store and recall any amount of information because of its secondary storage (a type of detachable memory) capability. It can retain a piece of information as long as a user desires and the user can recall the information whenever required. Even after several years, a user can recall exactly the same information that he/she had stored in the computer several years ago. A computer forgets or loses certain information only when a user asks it to do so. Hence, it is entirely up to the user to make a computer retain or forget some information.

7. No I. Q. - A computer is not a magical device. It possesses no intelligence of its own. Its I. Q. is zero, at least until today. It has to be told what to do and in what sequence. Hence, only a user determines what tasks a computer will perform. A computer cannot take its own decision in this regard.

8. No feelings- Computers are devoid of emotions. They have no feelings and no instincts because they are machines. Although men have succeeded in building a memory for computer, but no computer possesses the equivalent of a human heart and soul. Based on our feelings, taste, knowledge, and experience we often make certain judgements in our day-to-day life whereas, computers cannot make such judgements on their own. They make judgements based on the instructions given to them in the form of programs that are written by us (human beings).

Check your progress

Q1. Explain characteristics of computer.

1.4 Summary

In this unit you learnt about basics of computers like algorithms, types of algorithms, characteristic of algorithm, computer and elements of computer system and characteristic of computer.

- An algorithm is a set of rules for carrying out calculation either by hand or on a machine.
- Algorithmic is a branch of computer science that consists of designing and analyzing computer algorithms
- Each step of an algorithm must be **exact.**
- A computer is a programmable machine designed to perform arithmetic and logical operations automatically and sequentially on the input given by the user and gives the desired output after processing.
- Secondary storage provides large, non-volatile, and inexpensive storage for programs and data.

1.5 Review Questions

Q1. Define the term Algorithm. What is the benefit of using an algorithm?

Q2. Define the characteristic of an algorithm.

Q3. What is a computer? Define its elements and model with diagram.

Q4. Write short note on Input output device.

Q5. Define the characteristic of a computer.

UNIT-II

Problem Solving

Structure

2.0 Introduction

2.1 Information Representation.

2.2 Problem-solving Using Computers.

2.3 Summary

2.4 Review Questions

2.0 Introduction

In this unit we focused on problem solving. There are four main sections in this unit. In the first section i.e. Sec. 2.1 you will know about information representation. This section tells how information represent in the computer. In the Sec. 2.2 you will learn about problem solving using computer. Everybody can benefit from having good problem solving skills as we all encounter problems on a daily basis; some of these problems are obviously more severe or complex than others. It would be wonderful to have the ability to solve all problems efficiently and in a timely fashion without difficulty, unfortunately there is no one way in which all problems can be solved. So we can say problem solving consists of using generic or ad hoc methods, in an orderly manner, for finding solutions to problems. Some of the problem-solving techniques developed and used in artificial intelligence, computer science, engineering, mathematics, or medicine are related to mental problem-solving techniques studied in psychology. In Sec. 2.3 and 2.4 you will find summary and review questions respectively.

Objectives

After studying this unit you should be able to:

- Describe representation of information.
- Express problem solving using computers.
- Define flowcharts, pseudo codes etc.

2.1 Information Representation

Computer Programmers are problem solvers. In order to solve a problem on a computer you must:

1. Know how to **represent** the information (data) describing the problem.
2. Determine the steps to **transform** the information from one representation into another.

A computer, at heart, is dumb. It can only know about a few things... numbers, characters, Booleans, and lists (called arrays) of these items. Everything else must be "approximated" by combinations of these data types.

A good programmer will "encode" all the "facts" necessary to represent a problem in variables (See Variables). Further, there are "good ways" and "bad ways" to encode information. Good ways allow the computer to easily "compute" new information.

2.2 Problem solving using computers

Algorithm

An algorithm is a set of specific steps to solve a problem. Think of it this way: if you were to tell your 3 year old niece to play your favorite song on the piano (assuming the niece has never played a piano), you would have to tell her where the piano was, and how to sit on the bench, and how to open the cover, and which keys to press, and which order to press them in, etc.

The core of what good programmers do is being able to define the steps necessary to accomplish a goal. Unfortunately, a computer only knows a very restricted and limited set of possible steps. For example a computer can add two numbers. But if you want to find the average of two numbers, this is beyond the basic capabilities of a computer. To find the average, you must:

1. First: Add the two numbers and save this result in a variable
2. Then: Divide this new number the number two, and save this result in a variable.
3. Finally: provide this number to the rest of the program (or print it for the user).

Flowcharting

The second step in solving our problem involves the use of flowcharting. Flowcharting is a graphical way of depicting a problem in terms of its inputs, outputs, and processes. Though the shapes we will use in our flowcharts will be expanded as we cover more topics, the basic elements are as follows:

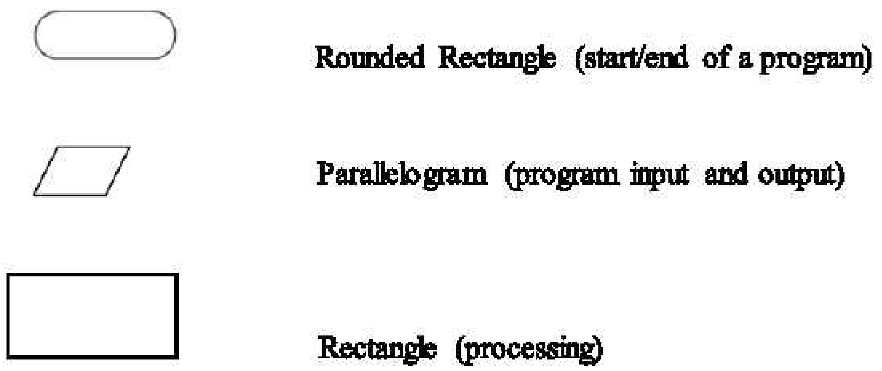

Rounded Rectangle (start/end of a program)
Parallelogram (program input and output)
Rectangle (processing)

Figure 1.6

The flowchart should proceed directly from the chart you developed in step one. First, lay out your starting node, as every one of your programs will have these.

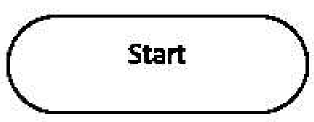


Figure 1.7

Next, begin adding your program elements sequentially, in the order that your problem description indicated. Connect the elements of your flowchart by uni-directional arrows that indicate the flow of your program.

According to our sample problem, we need to take in three items as input (length, width, and height) and after we have the user's input, need to process it. In this case, we must multiply the dimensions together.

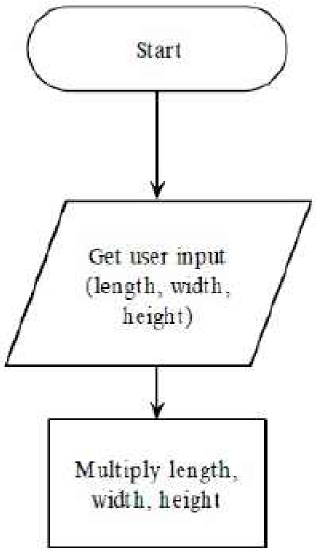

Start
Get user input (length, width, height)
Multiply length, width, height

Figure 1.8

Now, since our processing is complete, we should display the output for the user.

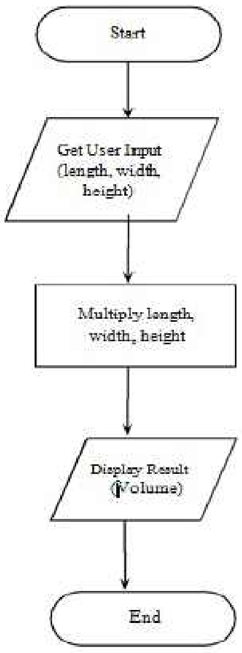

Start
Get User Input (length, width, height)
Multiply length, width, height
Display Result (Volume)
End

Figure 1.9

Pseudo code

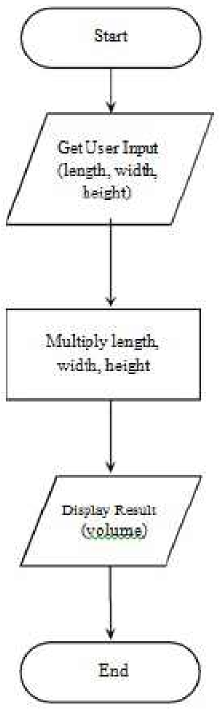
The final step in analyzing our problem is to step from our flowchart to **pseudo code.** Pseudo code

involves writing down all of the major steps you will use in the program as depicted in your flowchart. This is similar to writing final statements in your programming language without needing to worry about program syntax, but retaining the flexibility of program design.
Like flowcharting, there are many elements to pseudo code design; only the most rudimentary are described here.

Get	used to get information from the user
Display	used to display information for the user
Compute	perform an arithmetic operation
+	
-	
*	Standard arithmetic operators
/	
=	
( )	
Store	Store a piece of information for later use

It is important to note that each time you compute a value that you will need later, it is necessary to store it even if you will need it right away.

Here is the pseudo code for our example. It may be helpful to write out your pseudo code next to your flowchart


Start
Get User Input (length, width, height)
Multiply length, width, height
Display Result (volume)
End

```
Get length, width, height
Compute volume
        volume = length * width *
        height Store volume
Display volume
```

Figure 1.10

Check your progress

Q1. How information represent in computer?

Q2. Define flowchart and pseudo code.

2.3 Summary

In this unit you learnt about basics problem solving and you also learnt about the use of problem solving with the help of computer. The main elements of problem solving are algorithms, flowcharts, pseudo code etc.

- Computer Programmers are problem solvers.
- A good programmer will "encode" all the "facts" necessary to represent a problem in variables.
- An algorithm is a set of specific steps to solve a problem.
- Flowcharting is a graphical way of depicting a problem in terms of its inputs, outputs, and processes.
- Pseudo code involves writing down all of the major steps you will use in the program as depicted in your flowchart.

2.4 Review Questions

Q1. What do you mean by problem solving technique?

Q2. How computer helps in problem solving?

Q3. What do you mean by an algorithm?

Q4. What is flowchart? Define its elements with diagram.

Q5. Define pseudo code with example.

UNIT-III

Data Representation

Structure

3.0 Introduction

3.1 Representation of Characters in computers

3.2 Representation of Integers

3.3 Floating Point Number Representation

3.4 Summary

3.5 Review Questions

3.0 Introduction

In this unit we defined data representation. There are five sections in this unit. In the first section i.e. Sec. 3.1 we focused on representation of characters in computers. In Sec. 3.2 you will learn about representations of numbers i.e. integers. Computer uses *a fixed number of bits* to represent a piece of data, which could be a number, a character, or others. An *n*-bit storage location can represent up to 2^*n* distinct entities. For example, a 3-bit memory location can hold one of these eight binary patterns: 000, 001, 010, 011, 100, 101, 110, or 111. Hence, it can represent at most 8 distinct entities. You could use them to represent numbers 0 to 7, numbers 8881 to 8888, characters 'A' to 'H', or up to 8 kinds of fruits like apple, orange, banana; or up to 8 kinds of animals like lion, tiger, etc.

Integers, for example, can be represented in 8-bit, 16-bit, 32-bit or 64-bit. You, as the programmer, choose an appropriate bit-length for your integers. Your choice will impose constraint on the range of integers that can be represented. Besides the bit-length, an integer can be represented in various *representation* schemes, e.g., unsigned vs. signed integers. An 8-bit unsigned integer has a range of 0 to 255, while an 8-bit signed integer has a range of -128 to 127 - both representing 256 distinct numbers.

It is important to note that a computer memory location merely *stores a binary pattern*. It is entirely up to you, as the programmer, to decide on how these patterns are to be *interpreted*. For example, the 8-bit binary pattern "0100 0001B" can be interpreted as an unsigned integer 65, or an ASCII character 'A', or some secret information known only to you. In other words, you have to first decide how to represent a piece of data in a binary pattern before the binary patterns make sense. The interpretation of binary pattern is called *data representation* or *encoding*. Furthermore, it is important that the data representation schemes are agreed-upon by all the parties, i.e., industrial standards need to be formulated and straightly followed.

Once you decided on the data representation scheme, certain constraints, in particular, the precision and range will be imposed.

Hence, it is important to understand *data representation* to write *correct* and *high-performance* programs. In Sec.3.3 we define representation of fractions. In Sec. 3.4 and 3.5 you will find summary and review questions respectively.

Objectives

After studying this unit you should be able to:

- Describe representation of characters.
- Define representation of numbers.
- Express representation of integers, fractions.

3.1 Representation of Characters in computers

Even though many people used to think of computers as "number crunchers", people figured out long ago that it's just as important to handle character data.

Character data isn't just alphabetic characters, but also numeric characters, punctuation, spaces, etc. Most keys on the central part of the keyboard (except shift, caps lock) are characters.

As we've discussed with signed and unsigned ints, characters need to represent. In particular, they need to be represented in binary. After all, computers store and manipulate 0's and 1's (and even those 0's and 1's are just abstractions---the implementation is typically voltages).

Unsigned binary and two's complement are used to represent unsigned and signed int respectively, because they have nice mathematical properties, in particular, you can add and subtract as you'd expect.

However, there aren't such properties for character data, so assigning binary codes for characters is somewhat arbitrary. The most common character representation is ASCII, which stands for *American Standard Code for Information Interchange.*

There are two reasons to use ASCII. First, we need some way to represent characters as binary numbers (or, equivalently, as bit string patterns). There's not much choice about this since computers represent everything in binary.

If you've noticed a common theme, it's that we need representation schemes for everything. However, most importantly, we need representations for numbers and characters. Once you have that (and perhaps pointers), you can build up everything you need.

The other reason we use ASCII is because of the letter "S" in ASCII, which stands for "standard". Standards are good because they allow for common formats that everyone can agree on.

Unfortunately, there's also the letter "A", which stands for American. ASCII is clearly biased for the English language character set. Other languages may have their own character set, even though English dominates most of the computing world (at least, programming and software).

Nice Properties

Even though character sets don't have mathematical properties, there are some nice aspects about ASCII. In particular, the lowercase letters are contiguous ('a' through 'z' maps to 97_{10} through 122_{10}). The upper case letters are also contiguous ('A' through 'Z' maps to 65_{10} through 90_{10}). Finally, the digits are contiguous ('0' through '9' maps to 48_{10} through 57_{10}).

Since they are contiguous, it's usually easy to determine whether a character is lowercase or uppercase (by checking if the ASCII code lies in the range of lower or uppercase ASCII codes), or to determine if it's a digit, or to convert a digit in ASCII to an int value.

ASCII Code (Decimal)

0 nul	16 dle	32 sp	48 0	64 @	80 P	96 `	112 p
1 soh	17 dc1	33 !	49 1	65 A	81 Q	97 a	113 q
2 stx	18 dc2	34 "	50 2	66 B	82 R	98 b	114 r

3 etx	19 dc3	35 #	51 3	67 C	83 S	99 c	115 s
4 eot	20 dc4	36 $	52 4	68 D	84 T	100 d	116 t
5 enq	21 nak	37 %	53 5	69 E	85 U	101 e	117 u
6 ack	22 syn	38 &	54 6	70 F	86 V	102 f	118 v
7 bel	23 etb	39 '	55 7	71 G	87 W	103 g	119 w
8 bs	24 can	40 (	56 8	72 H	88 X	104 h	120 x
9 ht	25 em	41)	57 9	73 I	89 Y	105 i	121 y
10 nl	26 sub	42 *	58 :	74 J	90 Z	106 j	122 z
11 vt	27 esc	43 +	59 ;	75 K	91 [	107 k	123 {
12 np	28 fs	44 ,	60 <	76 L	92 \	108 l	124 |
13 cr	29 gs	45 -	61 =	77 M	93]	109 m	125 }
14 so	30 rs	46 .	62 >	78 N	94 ^	110 n	126 ~
15 si	31 us	47 /	63 ?	79 O	95 _	111 o	127 del

The characters between 0 and 31 are generally not printable (control characters, etc.). 32 is the space character.

Also note that there are only 128 ASCII characters. This means only 7 bits are required to represent an ASCII character. However, since the smallest size representation on most computers is a byte, a byte is used to store an ASCII character. The MSb of an ASCII character is 0.

Sometimes ASCII has been extended by using the MSb.

ASCII Code (Hex)

00 nul	10 dle	20 sp	30 0	40 @	50 P	60 `	70 p
01 soh	11 dc1	21 !	31 1	41 A	51 Q	61 a	71 q
02 stx	12 dc2	22 "	32 2	42 B	52 R	62 b	72 r
03 etx	13 dc3	23 #	33 3	43 C	53 S	63 c	73 s
04 eot	14 dc4	24 $	34 4	44 D	54 T	64 d	74 t
05 enq	15 nak	25 %	35 5	45 E	55 U	65 e	75 u
06 ack	16 syn	26 &	36 6	46 F	56 V	66 f	76 v
07 bel	17 etb	27 '	37 7	47 G	57 W	67 g	77 w
08 bs	18 can	28 (	38 8	48 H	58 X	68 h	78 x
09 ht	19 em	29)	39 9	49 I	59 Y	69 i	79 y
0a nl	1a sub	2a *	3a :	4a J	5a Z	6a j	7a z
0b vt	1b esc	2b +	3b ;	4b K	5b [	6b k	7b {
0c np	1c fs	2c ,	3c <	4c L	5c \	6c l	7c |

0d cr	1d gs	2d -	3d =	4d M	5d]	6d m	7d }
0e so	1e rs	2e .	3e >	4e N	5e ^	6e n	7e ~
0f si	1f us	2f /	3f ?	4f O	5f _	6f o	7f del

The difference in the ASCII code between an uppercase letter and its corresponding lowercase letter is 20_{16}.

This makes it easy to convert lower to uppercase (and back) in hex (or binary).

char as a one byte int

It turns out that C supports two **char** types: **char** (which is usually considered "signed") and **unsigned char**, which is unsigned.

This may seem like a particularly odd feature. What does it mean to have a signed or unsigned char?

This is where it's useful to think of a **char** as a one byte int. When you want to cast **char** to an **int** (of any size), the rules for sign-extension may apply. In particular, if the MSb of a **char** is 1, then casting it to an **int** may cause this 1 to sign extend, which may be surprising if you're not expecting it.

Of course, you may observe "but how would the MSb get the value 1?". If you recall, **char** is one of the data types that you can manipulate with bitwise and bit-shift operators. This means you can set or clear any bit of a **char**. In particular, you can set or clear the MSb of a **char**.

Another way you might get 1 for the MSb is casting an int down to a char. usually, this means truncating off the upper bytes, leaving the least significant byte. Since an int can have any bit pattern, there's a possibility that the least significant byte has a 1 in bit position b_7.

You should think of a **char** as both an ASCII character as well as an 8 bit int. This duality is important because **char** is the only data type that is 1 byte. There are no other data types that are 1 byte.

Other Character Codes

While ASCII is still popularly used, another character representation that was used (especially at IBM) was EBCDIC, which stands for *Extended Binary Coded Decimal Interchange Code* (yes, the word "code" appears twice). This character set has mostly disappeared. EBCDIC does not store characters contiguously, so this can create problems alphabetizing "words".

One problem with ASCII is that it's biased to the English language. This generally creates some problems. One common solution is for people in other countries to write programs in ASCII.

Other countries have used different solutions, in particular, using 8 bits to represent their alphabets, giving up to 256 letters, which is plenty for most alphabet based languages (recall you also need to represent digits, punctuation, etc.).

However, Asian languages, which are word-based, rather than character-based, often have more words than 8 bits can represent. In particular, 8 bits can only represent 256 words, which is far smaller than the number of words in natural languages.

Thus, a new character set called Unicode is now becoming more prevalent. This is a 16 bit code, which allows for about 65,000 different representations. This is enough to encode the popular Asian languages (Chinese, Korean, Japanese, etc.). It also turns out that ASCII codes are preserved. What does this mean? To convert ASCII to Unicode, take all one byte ASCII codes, and zero-extend them to 16 bits. That should be the Unicode version of the ASCII characters.

The biggest consequence of using Unicode from ASCII is that text files double in size. The second consequence is that endianness begins to matter again. With single bytes, there's no need to worry about endianness. However, you have to consider that with two byte quantities.

While C and C++ still primarily use ASCII, Java has already used Unicode. This means that Java must create a byte type, because **char** in Java is no longer a single byte. Instead, it's a 2 byte Unicode representation.

ASCII files

It's easy to fool yourself into thinking that numbers written in a file are actually the internal representation. For example, if you write 123 in a file using a text editor, is that really how the integer 123 is stored?

The file does NOT storing 123. Instead, it stores the ASCII code for the character '1', '2', and '3' (which is 31, 32, 33 in hex or 0011 0001, 0011 0010, 0011 0011 in binary).

ASCII files store bytes. Each byte is the ASCII code for some character in the character set. You can think of a text editor as a translator. It translates those binary numbers into symbols on the screen. Thus, when it sees41_{16}, that's the ASCII code for 'A', and thus 'A' gets displayed.

Some people think that if they type 0's and 1's in a text editor, they are writing out bits into a binary file. This is not true. The file contains the ASCII code for the *character* '0' and the *character* '1'.

There are hex editors which allow you to either type in binary or more commonly in hex. Those hex pairs are translated to binary. Thus, when you write F3, the binary number 1111 0011 is written to the file (the space is placed there only to make the binary number easy to read).

3.2 Representation of Integers

Integers are *whole numbers* or *fixed-point numbers* with the radix point *fixed* after the least-significant bit. They are contrast to *real numbers* or *floating-point numbers*, where the position of the radix point varies. It is important to take note that integers and floating-point numbers are treated differently in computers. They have different representation and are processed differently (e.g., floating-point numbers are processed in a so-called floating-point processor). Floating-point numbers will be discussed later.

Computers use *a fixed number of bits* to represent an integer. The commonly-used bit-lengths for integers are 8-bit, 16-bit, 32-bit or 64-bit. Besides bit-lengths, there are two representation schemes for integers:

1. *Unsigned Integers*: can represent zero and positive integers.
2. *Signed Integers*: can represent zero, positive and negative integers. Three representation schemes had been proposed for signed integers:
    a. Sign-Magnitude representation
    b. 1's Complement representation
    c. 2's Complement representation

You, as the programmer, need to decide on the bit-length and representation scheme for your integers, depending on your application's requirements. Suppose that you need a counter for counting a small quantity from 0 up to 200, you might choose the 8-bit unsigned integer scheme as there is no negative numbers involved.

3.2.1 n-bit Unsigned Integers

Unsigned integers can represent zero and positive integers, but not negative integers. The value of an unsigned integer is interpreted as "*the magnitude of its underlying binary pattern*".

Example 1: Suppose that *n*=8 and the binary pattern is 0100 0001B, the value of this unsigned integer is 1×2^0 + 1×2^6 = 65D.

An *n*-bit pattern can represent 2^*n* distinct integers. An *n*-bit unsigned integer can represent integers from 0 to (2^*n*)-1, as tabulated below:

n	**Minimum**	**Maximum**
8	**0**	(2^8)-1 (=255)
16	**0**	(2^16)-1 (=65,535)
32	**0**	(2^32)-1 (=4,294,967,295) (9+ digits)
64	**0**	(2^64)-1 (=18,446,744,073,709,551,615) (19+ digits)

Table 1

Signed Integers

Signed integers can represent zero, positive integers, as well as negative integers. Three representation schemes are available for signed integers:

1. Sign-Magnitude representation
2. 1's Complement representation
3. 2's Complement representation

In all the above three schemes, the *most-significant bit* (msb) is called the *sign bit*. The sign bit is used to represent the *sign* of the integer - with 0 for positive integers and 1 for negative integers. The *magnitude* of the integer, however, is interpreted differently in different schemes.

3.2.2 *n*-bit Sign Integers in Sign-Magnitude Representation

In sign-magnitude representation:

- The most-significant bit (msb) is the *sign bit*, with value of 0 representing positive integer and 1 representing negative integer.
- The remaining *n*-1 bits represent the magnitude (absolute value) of the integer. The absolute value of the integer is interpreted as "the magnitude of the (*n*-1)-bit binary pattern".

Example 1: Suppose that *n*=8 and the binary representation is 0 100 0001B.
Sign bit is 0 ⇒ positive

Absolute value is 100 0001B = 65D

Hence, the integer is +65D

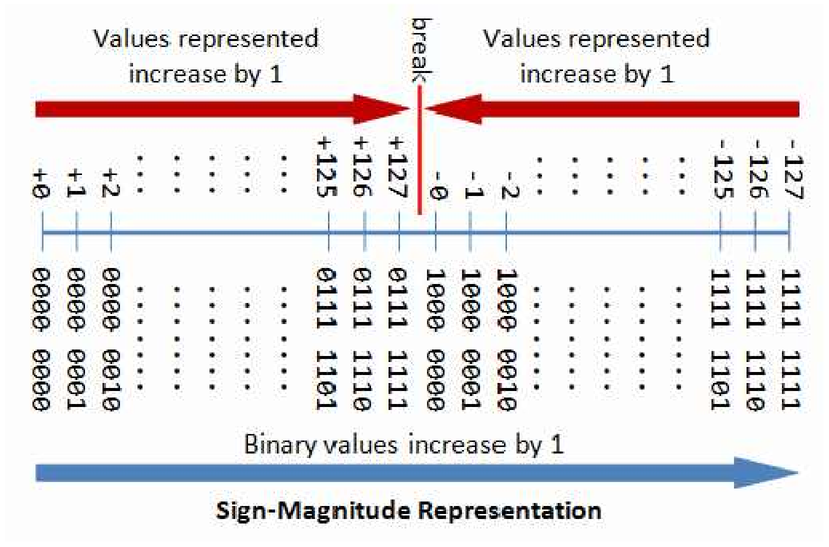


Figure 3.1

The drawbacks of sign-magnitude representation are:

1. There are two representations (0000 0000B and 1000 0000B) for the number zero, which could lead to inefficiency and confusion.
2. Positive and negative integers need to be processed separately.

3.2.3 n-bit Sign Integers in 1's Complement Representation

In 1's complement representation:

- Again, the most significant bit (msb) is the *sign bit*, with value of 0 representing positive integers and 1 representing negative integers.
- The remaining *n*-1 bits represents the magnitude of the integer, as follows:
  - For positive integers, the absolute value of the integer is equal to "the magnitude of the (*n*-1)-bit binary pattern".
  - For negative integers, the absolute value of the integer is equal to "the magnitude of the *complement* (*inverse*) of the (*n*-1)-bit binary pattern" (hence called 1's complement).

Example 1: Suppose that *n*=8 and the binary representation 0 100 0001B.
Sign bit is 0 ⇒ positive

Absolute value is 100 0001B = 65D

Hence, the integer is +65D

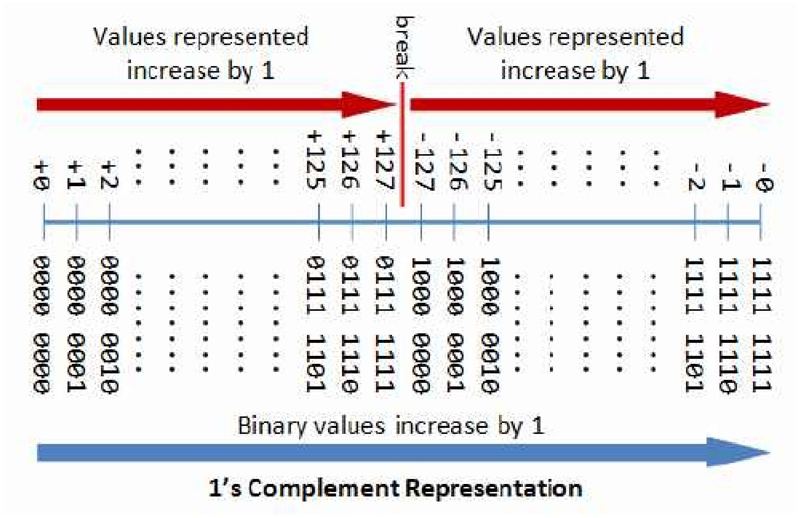


Figure 3.2

Again, the drawbacks are:

1. There are two representations (0000 0000B and 1111 1111B) for zero.
2. The positive integers and negative integers need to be processed separately.

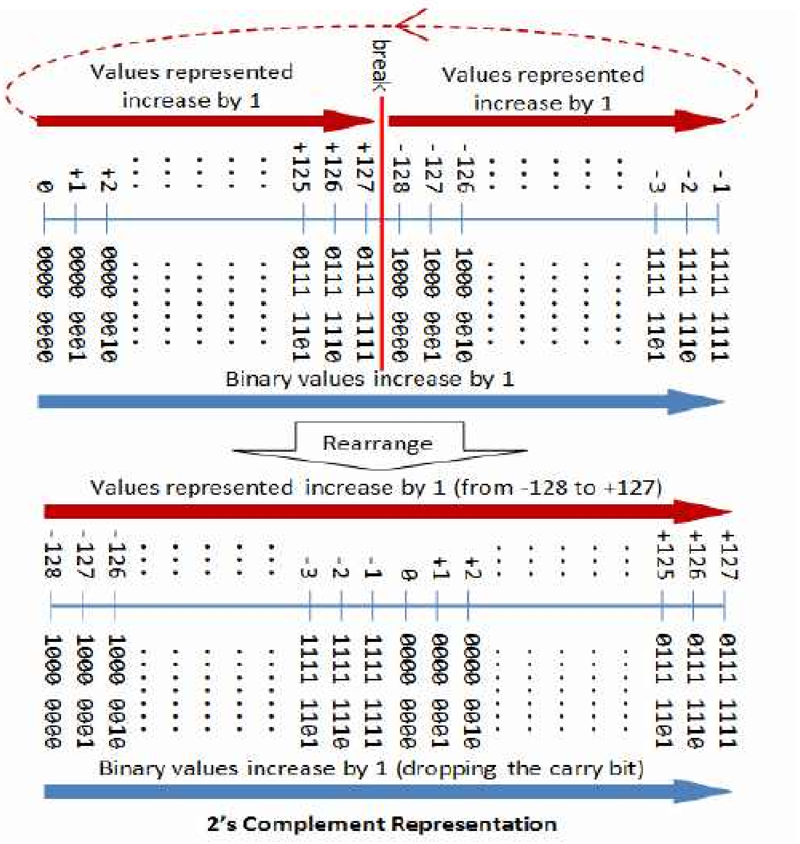
3.2.4 n-bit Sign Integers in 2's Complement Representation

In 2's complement representation:

- Again, the most significant bit (msb) is the *sign bit*, with value of 0 representing positive integers and 1 representing negative integers.
- The remaining *n*-1 bits represents the magnitude of the integer, as follows:
    - For positive integers, the absolute value of the integer is equal to "the magnitude of the (*n*-1)-bit binary pattern".
    - For negative integers, the absolute value of the integer is equal to "the magnitude of the *complement* of the (*n*-1)-bit binary pattern *plus one*" (hence called 2's complement).

Example 1: Suppose that *n*=8 and the binary representation 0 100 0001B.
Sign bit is 0 ⇒ positive
Absolute value is 100 0001B = 65D
Hence, the integer is +65D

Example 2: Suppose that *n*=8 and the binary representation 1 000 0001B.
Sign bit is 1 ⇒ negative
Absolute value is the complement of 000 0001B plus 1, i.e., 111 1110B + 1B = 127D
Hence, the integer is -127D



Values represented increase by 1
break
Values represented increase by 1
Binary values increase by 1
Rearrange
Values represented increase by 1 (from -128 to +127)
Binary values increase by 1 (dropping the carry bit)
2's Complement Representation

Figure 3.3

3.2.5 Computers use 2's Complement Representation for Signed Integers

We have discussed three representations for signed integers: signed-magnitude, 1's complement and 2's complement. Computers use 2's complement in representing signed integers. This is because:

1. There is only one representation for the number zero in 2's complement, instead of two representations in sign-magnitude and 1's complement.
2. Positive and negative integers can be treated together in addition and subtraction. Subtraction can be carried out using the "addition logic".

Example 1: Addition of Two Positive Integers: Suppose that n=8, 65D + 5D = 70D

65D → 0100 0001B

5D → 0000 0101B(+

0100 0110B → 70D (OK)

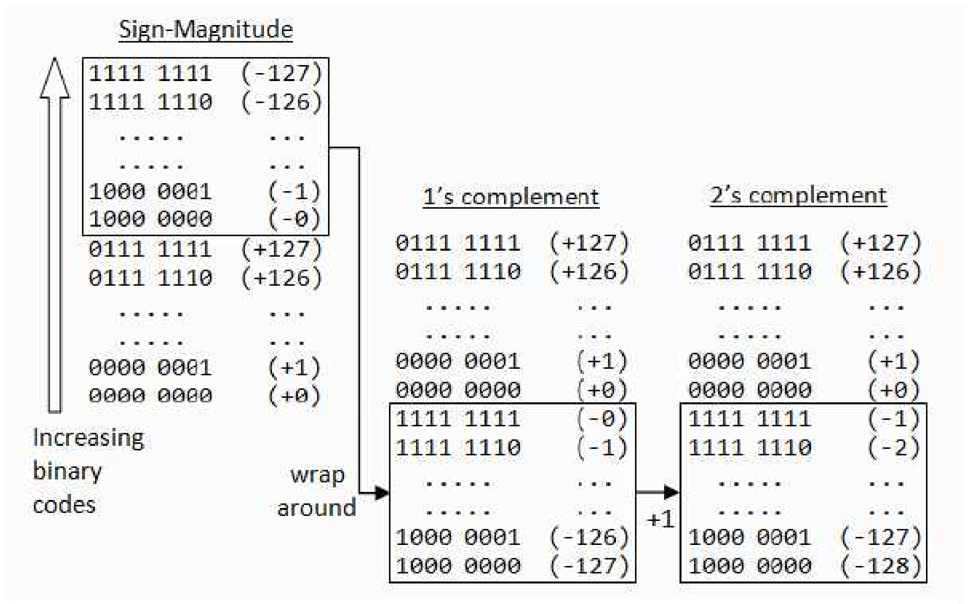


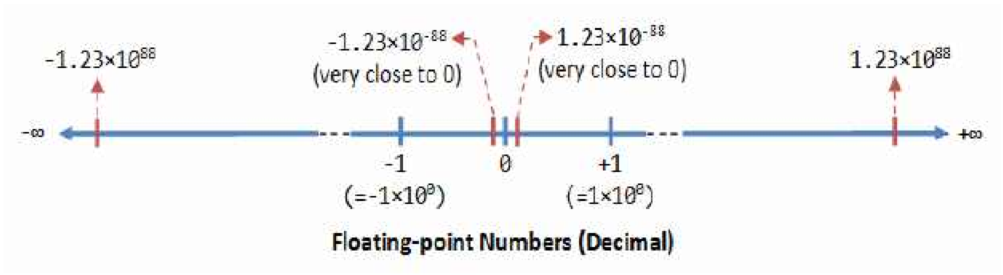
Figure 3.4

3.2.6 Decoding 2's Complement Numbers

1. Check the ***sign bit*** (denoted as S).
2. If S=0, the number is positive and its absolute value is the binary value of the remaining *n*-1 bits.
3. If S=1, the number is negative. You could "invert the *n*-1 bits and plus 1" to get the absolute value of negative number. Alternatively, you could scan the remaining *n*-1 bits from the right (least-significant bit). Look for the first occurrence of 1. Flip all the bits to the left of that first occurrence of 1. The flipped pattern gives the absolute value. For example,
4. n = 8, bit pattern = 1 100 0100B
5. S = 1 → negative
6. Scanning from the right and flip all the bits to the left of the first occurrence of 1 ⇒ 011 1100B = 60D
7. Hence, the value is -60D

3.3. Floating-Point Number Representation

A floating-point number (or real number) can represent a very large (1.23×10^88) or a very small (1.23×10^-88) value. It could also represent very large negative number (-1.23×10^88) and very small negative number (-1.23×10^88), as well as zero, as illustrated:



-1.23×10^88
-1.23×10^-88 (very close to 0)
1.23×10^-88 (very close to 0)
1.23×10^88
-∞
+∞
-1 (=-1×10^0)
0
+1 (=1×10^0)
Floating-point Numbers (Decimal)

Figure 3.5

A floating-point number is typically expressed in the scientific notation, with a *fraction* (F), and an *exponent* (E) of a certain *radix* (r), in the form of F×r^E. Decimal numbers use radix of 10 (F×10^E); while binary numbers use radix of 2 (F×2^E).

Representation of floating point number is not unique. For example, the number 55.66 can be represented as 5.566×10^1, 0.5566×10^2, 0.05566×10^3, and so on. The fractional part can be *normalized*. In the normalized form, there is only a single non-zero digit before the radix point. For example, decimal number 123.4567 can be normalized as 1.234567×10^2; binary number 1010.1011B can be normalized as 1.011011B×2^3.

It is important to note that floating-point numbers suffer from *loss of precision* when represented with a fixed number of bits (e.g., 32-bit or 64-bit). This is because there are *infinite* number of real numbers (even within a small range of says 0.0 to 0.1). On the other hand, a *n*-bit binary pattern can represent a *finite* 2^*n* distinct numbers. Hence, not all the real numbers can be represented. The nearest approximation will be used instead, resulted in loss of accuracy.

It is also important to note that floating number arithmetic is very much less efficient than integer arithmetic. It could be speed up with a so-called dedicated *floating-point co-processor*. Hence, use integers if your application does not require floating-point numbers.

In computers, floating-point numbers are represented in scientific notation of *fraction* (F) and *exponent* (E) with a *radix* of 2, in the form of F×2^E. Both E and F can be positive as well as negative. Modern computers adopt IEEE 754 standard for representing floating-point numbers. There are two representation schemes: 32-bit single-precision and 64-bit double-precision.

3.3.1 IEEE-754 32-bit Single-Precision Floating-Point Numbers

In 32-bit single-precision floating-point representation:

- The most significant bit is the *sign bit* (S), with 1 for negative numbers and 0 for positive numbers.
- The following 8 bits represent *exponent* (E).
- The remaining 23 bits represents *fraction* (F).

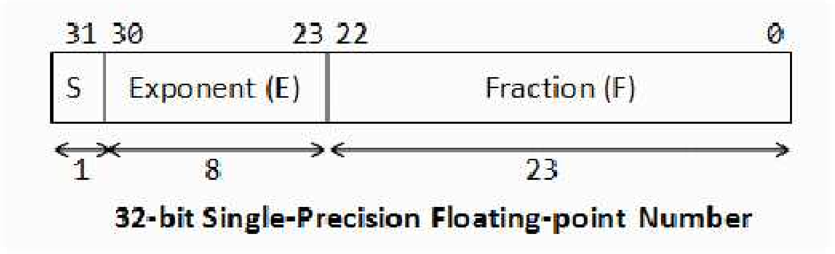


Figure 3.6

3.3.2 Normalized Form

Let's illustrate with an example, suppose that the 32-bit pattern is <u>1</u> <u>1000 0001</u> <u>011 0000 0000 0000 0000 0000</u>, with:

- S = 1
- E = 1000 0001
- F = 011 0000 0000 0000 0000 0000

In the *normalized form*, the actual fraction is normalized with an implicit leading 1 in the form of 1.F. In this example, the actual fraction is 1.011 0000 0000 0000 0000 0000 = 1 + 1×2^-2 + 1×2^-3 = 1.375D.

The sign bit represents the sign of the number, with S=0 for positive and S=1 for negative number. In this example with S=1, this is a negative number, i.e., -1.375D.

In normalized form, the actual exponent is E-127 (so-called excess-127 or bias-127). This is because we need to represent both positive and negative exponent. With an 8-bit E, ranging from 0 to 255, the excess-127 scheme could provide actual exponent of -127 to 128. In this example, E-127=129-127=2D.

Hence, the number represented is -1.375×2^2=-5.5D.

3.3.3 De-Normalized Form

Normalized form has a serious problem, with an implicit leading 1 for the fraction, it cannot represent the number zero! Convince yourself on this!

De-normalized form was devised to represent zero and other numbers.

For E=0, the numbers are in the de-normalized form. An implicit leading 0 (instead of 1) is used for the fraction; and the actual exponent is always -126. Hence, the number zero can be represented with E=0and F=0 (because 0.0×2^-126=0).

We can also represent very small positive and negative numbers in de-normalized form with E=0. For example, if S=1, E=0, and F=011 0000 0000 0000 0000 0000. The actual fraction is0.011=1×2^-2+1×2^-3=0.375D. Since S=1, it is a negative number. With E=0, the actual exponent is -126. Hence the number is -0.375×2^-126 = -4.4×10^-39, which is an extremely small negative number (close to zero).

Check your progress

Q1. How integer number represent in computer?

Q2. How character represent in computer?

Q3. How fractional number represent in computer?

3.4 Summary

In this unit you learnt representation of data in computer, representation of integers in computer, representation of fractions.

- Computer uses *a fixed number of bits* to represent a piece of data, which could be a number, a character, or others.
- The interpretation of binary pattern is called *data representation* or *encoding*.
- Character data isn't just alphabetic characters, but also numeric characters, punctuation, spaces, etc.
- We use ASCII is because of the letter "S" in ASCII, which stands for "standard".
- Integers are *whole numbers* or *fixed-point numbers* with the radix point *fixed* after the least-significant bit.

3.5 Review Questions

Q1. What is data representation? Explain with example.

Q2. How an integer store in computers memory? Justify your answer.

Q3. Define the working of fractional representation of numbers in computer.

Q4. Define 1's compliment with example.

Q5. Define 2's complement with example.

UNIT-IV

Hexadecimal Representation

Structure

4.0 Introduction
4.1 Hexadecimal Representation of Numbers
4.2 Decimal to Binary Conversion
4.3 Error-detecting codes
4.4 Summary
4.5 Review Questions

4.0 Introduction

In this unit we define representation of hexadecimal Numbering System. In this unit there are five sections. In the first section i.e. Sec. 4.1 we explained hexadecimal representation of numbers. As you already know the one main disadvantage of binary numbers is that the binary string equivalent of a large decimal base-10 number can be quite long. When working with large digital systems, such as computers, it is common to find binary numbers consisting of 8, 16 and even 32 digits which makes it difficult to both read and write without producing errors especially when working with lots of 16 or 32-bit binary numbers. One common way of overcoming this problem is to arrange the binary numbers into groups or sets of four bits (4-bits). These groups of 4-bits uses another type of numbering system also commonly used in computer and digital systems called Hexadecimal Numbers. The "Hexadecimal" or simply "Hex" numbering system uses the Base of 16 system and are a popular choice for representing long binary values because their format is quite compact and much easier to understand compared to the long binary strings of 1's and 0's. Being a Base-16 system, the hexadecimal numbering system therefore uses 16 (sixteen) different digits with a combination of numbers from 0 through to 15. In other words, there are 16 possible digit symbols. However, there is a potential problem with using this method of digit notation caused by the fact that the decimal numerals of 10, 11, 12, 13, 14 and 15 are normally written using two adjacent symbols. For example, if we write 10 in hexadecimal, do we mean the decimal number ten, or the binary number of two (1 + 0). To get around this tricky problem hexadecimal numbers that identify the values of ten, eleven, . . . , fifteen are replaced with capital letters of A, B, C, D, E and F respectively.

Then in the Hexadecimal Numbering System we use the numbers from 0 to 9 and the capital letters A to F to represent its Binary or Decimal number equivalent, starting with the least significant digit at the right hand side.

As we have just said, binary strings can be quite long and difficult to read, but we can make life easier by splitting these large binary numbers up into even groups to make them much easier to write down and understand. For example, the following group of binary digits 1101 0101 1100 11112 are much easier to read and understand than 11010101110011112 when they are all bunched up together.

In the everyday use of the decimal numbering system we use groups of three digits or 000's from the right hand side to make a very large number such as a million or trillion, easier for us to understand and the same is also true in digital systems.

Hexadecimal Numbers is a more complex system than using just binary or decimal and is mainly used when dealing with computers and memory address locations. By dividing a binary number up into groups of 4 bits, each group or set of 4 digits can now have a possible value of between "0000" (0) and "1111" (8+4+2+1 = 15) giving a total of 16 different number combinations from 0 to 15. Don't forget that "0" is also a valid digit. In Sec. 4.2 you will know decimal to binary conversion, in

Sec. 4.3 you will learn about error detection codes. In Sec. 4.4 and 4.5 you will find summary and review questions respectively.

Objectives

After studying this unit you should be able to:

- Describe hexadecimal representation of numbers.
- Define conversion between numbers.
- Express error detection codes.

4.1 Hexadecimal Representation of Numbers

Hexadecimal is a base-16 number system. It is a different method of representing numbers than thebase-10 system we use in every day practice. In base-10, we count in multiples of 10 before adding another digit. For example, "8 - 9 - 10 - 11 - 12..." and "98 - 99 - 100 - 101 - 102" Notice how a new digit is added when the number 10 is reached, and another digit is added to represent 100 (10x10). In base-16, or the hexadecimal number system, each digit can have sixteen values instead of ten. The values of a hexadecimal digit can be:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Therefore, the number 12 (in the common base-10 format) would be represented as "C" in hexadecimal notation. The number 24 would be 18 (16+8). 100 is 64 in hexadecimal (16x6 + 4) and 1000 is 3E8 (256x3 + 16x14 + 8).

While computers process numbers using the base-2 or binary system, it is often more efficient to visually represent the numbers in hexadecimal format, this is because it only takes one hexadecimal digit to represent four binary digits. Since there are eight binary digits in a byte, only two hexadecimal digits are needed to represent one byte.

Hexadecimal Representation of Selected Numbers

Value	**float**	**double**	**long double**
+0 -0	00000000 80000000	0000000000000000 8000000000000000	00000000000000000000000000000000 80000000000000000000000000000000
+1.0 -1.0	3F800000 BF800000	3FF0000000000000 BFF0000000000000	3FFF0000000000000000000000000000 BFFF0000000000000000000000000000
+2.0 +3.0	40000000 40400000	4000000000000000 4008000000000000	40000000000000000000000000000000 40080000000000000000000000000000
+Infinity -Infinity	7F800000 FF800000	7FF0000000000000 FFF0000000000000	7FFF0000000000000000000000000000 FFFF0000000000000000000000000000
NaN	7FBFFFFF	7FF7FFFFFFFFFFFF	7FFF7FFFFFFFFFFFFFFFFFFFFFFFFFFF

Table -1 Hexadecimal Representation of Selected Numbers (SPARC)

Value	float	double	long double
+0 -0	00000000 80000000	0000000000000000 0000000080000000	00000000000000000000 80000000000000000000
+1.0 -1.0	3F800000 BF800000	000000003FF00000 00000000BFF00000	3FFF8000000000000000 BFFF8000000000000000
+2.0 +3.0	40000000 40400000	0000000040000000 0000000040080000	40008000000000000000 4000C000000000000000
+Infinity -Infinity	7F800000 FF800000	000000007FF00000 00000000FFF00000	7FFF8000000000000000 FFFF8000000000000000
NaN	7FBFFFFF	FFFFFFFF7FF7FFFF	7FFFBFFFFFFFFFFFFFFF

Table-2 Hexadecimal Representation of Selected Numbers (x86)

Hexadecimal to Binary conversion

Replace each digit by a set of four binary digits and group these together.

Example1.12 Convert $(765.3)_{16}$ into binary

Sol

7 6 5 . 3

=0111 0110 0101 .0 011

=011101100101.0011

Thus the binary equivalent of 756.3 is 011101100101.0011

Converting between octal and hexadecimal

The method involves following steps:

1. Convert each octal to 3-bit binary form
2. Combine all the 3-bit binary numbers
3. Segregate the binary numbers into 4- bit binary form by starting the first number from right bit(LSB) towards the number on the left bit(MSB)
4. Finally, convert these 4-bit blocks into their respective hexadecimal symbols.

Example 1.13 Determine the hexadecimal equivalent of $(2327)_8$

Octal Number	2	3	2	7
Binary Value	010	011	010	111
Combine all the 3-bit binary values	0100 1101 0111			
Separating the groups of binary numbers into the 4-bit binary number	0100	1101	0111	
Hexadecimal Equivalent	4	D	7	

Table-2

Thus hexadecimal equivalent of $(2327)_8$ is $(4D7)_{16}$

Example 1.14 Determine octal equivalent of $(2B6)_{16}$

Hexadecimal Number	2	B	6	
Binary Value	0010	1011	0110	
Combine all the 4-bit binary values	0010 1011 0110			
Separating the groups of binary numbers into the 4-bit binary number	001	010	110	110
Octal Equivalent	1	2	6	6

Table-3

Thus, octal equivalent of $(2B6)_{16}$ is $(1266)_8$

4.2 DECIMAL to BINARY Conversion

When converting from decimal to binary the mathematical way is simplest. Start with the decimal number you want to convert and start dividing that number by two, keeping track of the remainder after each complete division. Every time you divide by two, you will divide evenly (0) or get a remainder of one (1). Following the pattern to the end, you will get a binary number. Write the remainders in the order they were generated from right to left and the result is the equivalent binary value.

Example: Convert decimal 44 to binary.

DIVIDE

44 / 2 = 22 remainder = **0**

22 / 2 = 11 remainder = **0**

11 / 2 = 5 remainder = **1**

5 / 2 = 2 remainder = **1**

2 / 2 = 1 remainder = **0**

1 / 2 = 0 remainder = **1**

REVERSE THE ORDER OF REMAINDERS

The bits, in the order they were generated is 001101 Reversing the order of bits we get 101100. Properly padded with leading zeroes to fill out one byte, we get 01011000

4.3 Error Detection Codes

- Transmitted binary information is subject to noise that could change bits 1 to 0 and vice versa
- An error detection code is a binary code that detects digital errors during transmission.
- The detected errors cannot be corrected, but can prompt the data to be retransmitted.
- The most common error detection code used is the parity bit.
- A parity bit is an extra bit included with a binary message to make the total number of 1's either odd or even

TABLE — Parity Bit Generation

Message *xyz*	*P*(odd)	*P*(even)
000	1	0
001	0	1
010	0	1
011	1	0
100	0	1
101	1	0
110	1	0
111	0	1

Table-4

The P (odd) bit is chosen to make the sum of 1's in all four bits odd

- The even-parity scheme has the disadvantage of having a bit combination of all 0's
- Procedure during transmission:
  - At the sending end, the message is applied to a parity generator
  - The message, including the parity bit, is transmitted
  - At the receiving end, all the incoming bits are applied to a parity checker
  - Any odd number of errors are detected
- Parity generators and checkers are constructed with XOR gates (odd function).
- An odd function generates 1 iff an odd number if input variables are 1

Figure ▬ Error detection with odd parity bit.

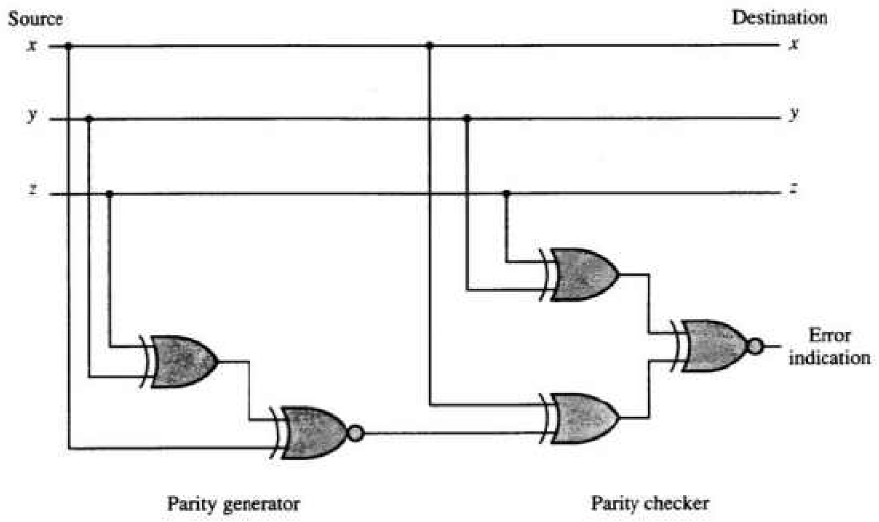


Figure 4-1

Check your progress

Q1. Convert (673.5)16 into binary

Q2. Determine octal equivalent of (1A5)16

Q3. Convert decimal 64 to binary.

4.4 Summary

In this unit you learnt about detail knowledge of hexadecimal data representation in computer memory. You also learnt about the decimal to binary conversion and error detection codes.

- Hexadecimal is a base-16 number system.
- In base-10, we count in multiples of 10 before adding another digit.
- Hexadecimal number system, each digit can have sixteen values instead of ten.
- An error detection code is a binary code that detects digital errors during transmission.
- Parity generators and checkers are constructed with XOR gates (odd function)

4.5 Review Questions

Q1. Define hexadecimal number system in detail.

Q2. Explain hexadecimal representation of numbers in computers memory.

Q3. How a decimal-to-binary conversion takes place? Justify your answer.

Q4. Write a short note on 1's and 2's complement.

Q5. What is error detection code? How computer detect the error and correct the code? Elaborate your answer.

BIBLIOGRAPHY

Pradeep K. Sinha, Priti Sinha (2004). *Computer fundamentals.* BPB

Larry Long. (2004). *Computer Fundamentals.* New Delhi: Dream Tech Press.

Bissmer, R.H. (1993). *Introduction to Computer Concepts.* New York: John Wiley.

Rajaraman, V (1997). *Fundamentals of Computers. 2nd edition.* New Delhi: Prentice-Hall of India.

Anderson, R.G. (1900). *Data Processing (Vol. 1: Principles and practice; Vol 2: Information Systems and Technology),* London: Pitman.

Ramesh Bangia. *Computer Fundamentals and Information Technology*- FIREWALL Media

Ceruzzi, P.E. (1998). *A history of modern computing.* Cambridge, M.A.: MIT Press.

Jain, Satish (1999). *Information technology.* New Delhi: BPB Publications.

Jaiswal, S. (2000). *Information technology today.* New Delhi: Galgotia Pub.

Mahapatra, M and Ramesh D.B. (eds.) (2004), *Information technology application in libraries: a textbook for beginners.* Bhubaneswar: Reprint.

Droomy, R.G. (2004). *How to solve it by computer.* New Delhi: Prentice Hall of India.

O'Leary, Timothy J. and O'Leary, Linda I. (2002). *Computing Essentials 2002-2003.* International Edition. New York: McGraw-Hill Irwin.

Pratt, Terrence W. and Zelkowitz, Marvin V. (2003). *Programming languages: design and implementation.* 4th Edition. New Delhi: Pearson Education.

Schneider, G. Michael and Gersting, Judith L. (1998). *An invitation to computer science.* 2nd Edition. California: Brooks/Cole Publishing Company.

Williams, Brian K. and Sawyer, Stacey C.. (2003). *Using information technology: a practical introduction to computers and communications.* 5th Edition. New Delhi: Tata McGraw-Hill Publishing.

Uttar Pradesh Rajarshi Tandon Open University

MBA -3.51

Computer Fundamentals and its Organisation

BLOCK

2

Peripheral Devices and Memory

Unit-I 51

Input & Output Devices

Unit-II 72

Computer Memory

Unit-III 93

Memory In Physical Devices

Unit-IV 109

Disk Drives

Printed and Published by Prof. P.P. Dubey, Registrar, Uttar Pradesh Rajarshi Tandon Open University , Prayagraj, 2022.

Printed By : K.C.Printing & Allied Works, Panchwati, Mathura -281003.

Block - 2 : Peripheral Devices and Memory

This is the second block of this course on peripheral devices and memory. Peripheral devices plays a very important role in computer. We can input with a number of peripheral devices in to the computer and we can get output with the help of these peripheral devices. In the same manner memory also play a huge role in computer. This block has detailed information about the peripheral devices and memory as well and having four following units.

In the first unit we focused on input & output devices. There are number of input and output devices. With the help of these devices we can input the data and we also can get output. These input devices are Keyboard, Pointing Devices, Speech Recognition, Digital Camera, Scanners, and Other Input Methods Optical Scanners. Similarly the output devices are monitor, printer, plotters etc. We also focused on the description of computer input & output units.

In the second unit we discussed about the computer memory. As we all know the memory is very important part if the computer. This unit has the detailed description of the memory. In this unit we described about Memory Cells, Memory Cell Address, Memory Organization, Register Memory, Main Memory (RAM), Disk Memory, SRAM, DRAM, SDRAM, DDR SDRAM, ROM, PROM, EPROM, EEPROM, Flash, Main Memory Organization, Byte Addressing, Byte Ordering, Memory Modules, Memory Chips, and Interleaved Memory. We also described about the Read-only memory and its types like PROM - Programmable Read Only Memory, EPROM - Erasable Programmable Read Only Memory, EEPROM - Electrically Erasable Programmable Read Only Memory, and Flash EEPROM memory.

In the third unit we told about the RAM. We also told about the where the RAM resides in computer and how we can add the RAM in computer. We also define the types of RAM. This unit also contains physical devices and its use to construct memories. We also discussed about the magnetic hard disk.

In the fourth unit we highlight the concept of floppy disk drives, Parts of the Floppy Disk, Protective Components, Housing, Shutter and spring, Recording Components, Magnetic Disk. We also define the working of floppy disk, CD-ROM compact disc read only memory, CD-ROM sectors & modes, capacity of a CD-ROM, magnetic tape drives, classification and application of tape drives, compact disk read only memory and magnetic tape drives.

As you study the material you will come across abbreviations in the text, e.g. Sec. 1.1, Eq.(1 .l) etc. The abbreviation Sec. stands for section and Eq. for equation. Figure, a. b refers to the bth figure of Unit a, i.e. Figure. 1.1 is the first figure in Unit 1. Similarly, Sec. 1.1 is the first section in Unit 1 and Eq.($.8) is the eighth equation in Unit 4. Similarly Table x. y refers to the yth table of Unit x, i.e. Table. 1.1 is the first table in Unit 1.

In your study you will also find that the units are not of equal-length and your study time for each unit will vary.

We hope you enjoy studying the material and wish you success.

UNIT-I

Input & Output Devices

Structure

1.0 Introduction

1.1 Input & Output Devices

1.2 Description of Computer Input Units

1.3 Computer Output Units

1.4 Summary

1.5 Review Questions

1.0 Introduction

This is the first unit of this block. In this unit we introduce the concept of Input and Output devices. In this unit there are five sections. In Sec. 1.1 you will learn about input output devices. As you have studied earlier Input output devices also known as peripheral, computer peripheral, used to enter information and instructions into a computer for storage or processing and to deliver the processed data to a human operator or, in some cases, a machine controlled by the computer. Such devices make up the peripheral equipment of modern digital computer systems. Peripherals are commonly divided into three kinds: input devices, output devices, and storage devices (which partake of the characteristics of the first two). An input device converts incoming data and instructions into a pattern of electrical signals in binary code that are comprehensible to a digital computer. An output device reverses the process, translating the digitized signals into a form intelligible to the user. At one time punched-card and paper-tape readers were extensively used for inputting, but these have now been supplanted by more efficient devices. In Sec. 1.2 you will know about description of computer input unit. In Sec. 1.3 some other input devices has been introduced. In Sec. 1.4 we focused on computer output devices. In Sec. 1.5 and 1.6 you will find summary and review questions respectively.

Objectives

After studying this unit you should be able to:

1. Define input, output devices.
2. Express description of input and output devices.

1.1 Input & Output Devices:

A computer accepts (input) information and manipulates (processes) it to get desired result (output) on a sequence of instructions. In the previous lesson, we discussed that a computer system essentially consists of three components: *input devices, central processing unit,* and *output devices.*

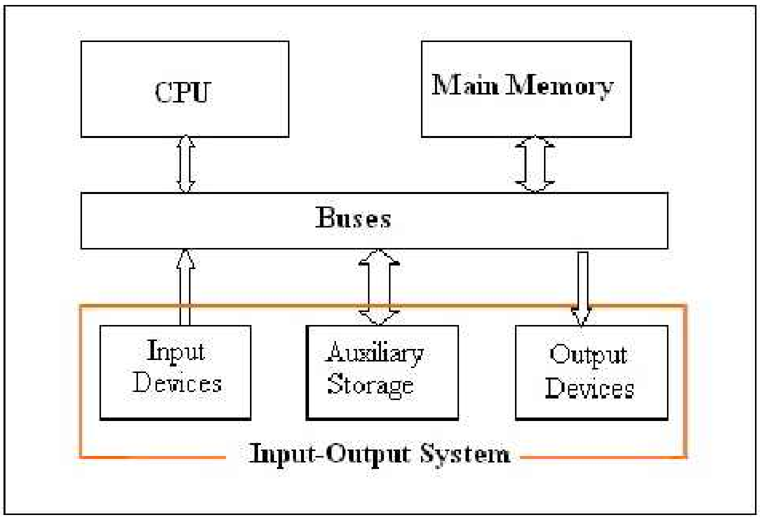


Figure 1-1

Input devices are used to provide data to the central processing unit for processing. The aim of this lesson is to familiarize you with the various types of input devices along with their advantages, disadvantages, and applications. An output device is any peripheral device that converts machine-readable information into people-readable form such as a monitor, printer, plotter and voice output device.

Introduction

Input devices are used to provide data to the central processing unit for processing. After processing, the input data is converted into meaningful information and this output is presented to the user with the help of output devices. In computer terminology devices can be refereed as a unit of hardware, which is capable of providing input to the computer or receiving output or both. An input device captures information and translates into form understandable by computer and output devices translate information into form understandable by human-being as shown in fig 1-2. Input devices let the user talk to the computer. Output devices let the computer communicate to the user. The common input devices are *keyboards* and *mouse*. The output devices are *monitors* and *printers*

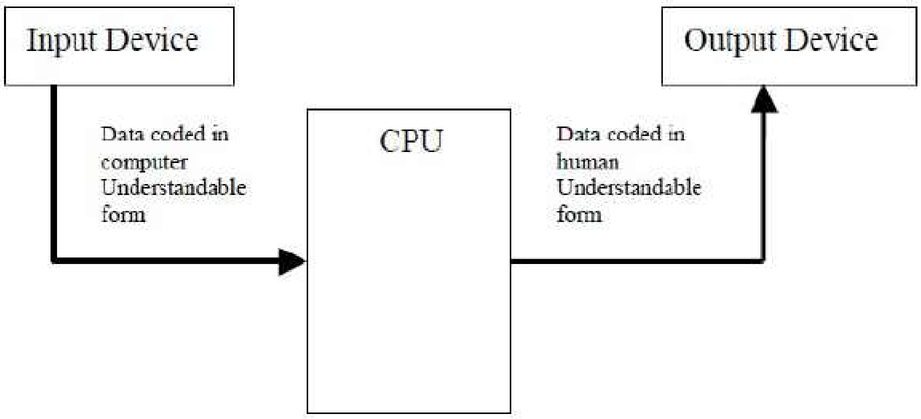

Input Device
Output Device
Data coded in computer Understandable form
CPU
Data coded in human Understandable form

Figure1-2

1.2 Description of Computer Input Units

Input devices can be broadly classified into the following categories:

- Keyboard
- Pointing Devices
- Speech Recognition
- Digital Camera
- Scanners
- Other Input Methods (Optical Scanners)

Keyboard

Keyboard is designed to resemble a regular typewriter with a few additional keys. A keyboard is the most common data entry device. Using a keyboard, the user can type text and execute commands. Data is entered into computer by simply pressing various keys. The layout of a keyboard come in various styles such as QWERTY, DVORAK, AZERTY but the most common layout is the QWERTY. It is named so because the first six keys on the top row of letters are *Q,W E, R, T,* and *Y.* The number of keys on a typical keyboard varies from 82 keys to 108 keys. Portable computers such as laptops quite often have custom keyboards that have slightly different key arrangements than a standard keyboard. In addition, many system manufacturers add special buttons to the standard layout. Keyboard is the easiest input device, as it does not require any special skill, it is supplied with a computer so no additional cost is incurred. The maintenance and operation cost of keyboard is also less. However, using a keyboard for data entry may be a slow process.

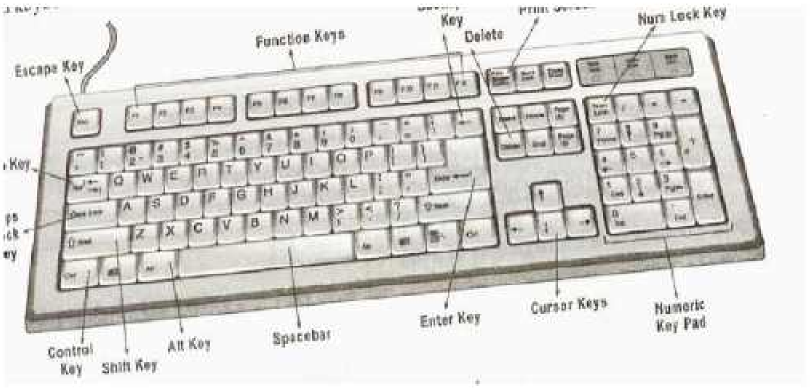


Figure 1-3

The layout of the keyboard can be divided into the following five sections: **Typing Keys:** These keys include the letter keys (1, 2, A, B, etc.), which are generally laid out in the same style that was common for typewriters.

Numeric Keypad: Numeric keys are located on the right hand side of the keyboard. Generally, it consists of a set of 17 keys that are laid out in the same configuration used by most adding machines and calculators.

Function Keys: The functions keys (FI, F2, F3, etc.) are arranged in a row along the top of the keyboard and could be assigned specific commands by the current application or the operating system.

Control Keys: These keys provide cursor and screen control. It includes four directional arrows (← ↑ → ↓). These keys allow the user to move the cursor on the display area one space at a time in either an up, down, left or right direction. Control keys also include *Home, End, Insert, Delete, Page Up, Page Down, Control (Ctrl), Alternate (A It),* and *Escape (Esc).*

Special Purpose Keys: Apart from the above-mentioned keys, a keyboard contains some special purpose keys such as *Enter, Shift, Caps Lock, Num Lock, Spacebar, Tab,* and *Print Screen.*

Working of a Keyboard

A keyboard is a series of switches connected to a small keyboard microprocessor. When the user presses a key, it causes a change in the amount of current flowing through the circuit associated specifically with that key. The keyboard microprocessor detects this change in current flow. By doing this, the processor can tell when a key has been pressed and when it is being released. The processor generates the associative code, known as *scan code,* of the key and sends it to the operating system. A copy of this code is also stored in the keyboard's memory.

Pointing Devices

In some applications, keyboard is not convenient. For example, if the user wants to select an item from a list, the user can identify that items position by selecting it through the keyboard. However, this action could be performed quickly by pointing at correct position. A *pointing device* is used to communicate with the computer by pointing to location on the screen. Some of the commonly used pointing devices are **mouse, trackball, joystick, light pen, and touch panel.**

Mouse

Mouse is a small hand-held pointing device, which is rectangular-shaped with a rubber ball embedded at its lower side and buttons on the top. Usually a mouse contains two or three buttons, which can be used to input commands or information. Figure 3.3 shows a mouse with three buttons.

Figure 1-4

The mouse may be classified as a *mechanical* mouse or an *optical* mouse, based on technology it uses.

A *mechanical mouse* uses a rubber ball at the bottom surface, which rotates as the mouse is moved along a flat surface, to move the cursor. Mechanical mouse is the most common and least expensive pointing device. Microsoft, IBM, and Logitech are some well-known makers of the mechanical mouse.

An *optical mouse* uses a light beam instead of a rotating ball to detect movement across a specially patterned mouse pad. As the user rolls the mouse on a flat surface, the cursor on the screen also moves in the direction of the mouse's movement.

An optical mouse has the following benefits over the mechanical mouse:

- No moving part means less wear and a lower chance of failure.
- Dirt cannot get inside the mouse and hence no interference with the tracking sensors.
- They do not require a special surface such as a mouse pad.

The cursor of the mouse can be *text cursor graphic cursor*. The text cursor (I) is used for text while the graphic cursor (↖) is used for pointing and drawing.

A mouse allows us to create graphic elements on the screen, such as lines, curves, and freehand shapes. Since it is an intuitive device, it is much easier and convenient to work as compared to the keyboard. Like keyboard, usually it is also supplied with a computer; therefore, no additional cost is incurred. The mouse cannot easily be used with laptop, notebook or palmtop computers. These need a track ball or a touch sensitive pad called a touch pad

Working of a mouse

A mechanical mouse has a rubber ball in the bottom. When the user moves the mouse, the ball rolls along the surface of the mouse pad, and the mouse keeps track of how far the ball rolls. This allows it to tell how far it has moved. Inside the bottom of the mouse are three rollers. These rollers are mounted at a 90° angle to the one other, one roller measures how fast the ball is turning horizontally, and the other measures how fast it is turning vertically. When the ball rolls, it turns these two rollers. The rollers are connected to axles, and the axles are connected to a small sensor that measures how fast the axle is turning. Both sets of information are passed to the electronics inside the mouse. This little processor, usually consisting of little more than a single chip, uses the information to determine how fast the mouse itself is Processor Chip moving, and in what direction. This information is passed to the computer via mouse cord, where the operating system then moves the pointer accordingly.

The optical mouse uses an infrared light and special mouse pads with fine grid lines to measure the rotation of the axle. The axle in optical mouse is connected to a little photo-interrupter wheel with a number of tiny holes in it. In front of this wheel is a light and on the other side of the wheel is a light meter. As the wheel turns, the light flashes through the holes in the wheel. By measuring how often these flashes occur, the light sensor can measure how fast the wheel is turning and sends the corresponding coordinates to the computer. The computer moves the cursor on the screen based on the coordinates received from the mouse. This happens hundreds of times each second, making the cursor appear to move very smoothly.

Trackball

Trackball is another pointing device that resembles a ball nestled in a square cradle and serves as an alternative to a mouse. In general, a trackball is as if a mouse is turned upside down. It has a ball, which can be rotated by fingers in any direction, the cursor moves accordingly. The size of the ball of the trackball varies from as large as a cue ball, to as small as a marble. Since, it is a static device so rather than rolling the mouse on the top of the table, the ball on the top is moved by using fingers, thumbs, and palms.

This pointing device comes in various shapes and forms but with the same functionality. The three shapes, which are commonly used, are a ball, a button, and a square.

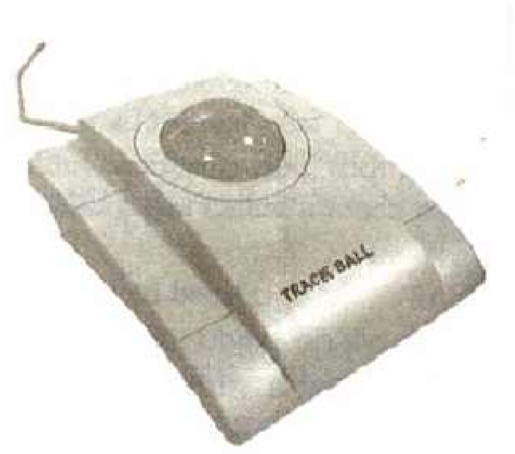


Figure 1-5

Joystick

Joystick is a device that moves in all directions and controls the movement of the cursor. The joystick offers three types of control: *digital, glide,* and *direct.*

Digital control allows movement in a limited number of directions such as up, down, left, and right.

Glide and direct control allow movements in all directions (360 degrees). Direct control joysticks have the added ability to respond to the distance and speed which user moves the stick.

A joystick is generally used to control the velocity of the screen cursor movement rather than its absolute position. Joysticks are mainly used for computer games, for other applications, which includes flight simulators, training simulators, CAD/CAM systems, and for controlling industrial robots.

Figure 1-6

Light Pen

It is the pen like device, which is connected to the machine by a cable. A light pen (sometimes called a mouse pen) is a hand-held electro-optical pointing device which when touched to or aimed closely at a connected computer monitor, will allow the computer to determine where on that screen the pen is aimed. It actually does not emit light; its light sensitive-diode would sense the light coming from the screen. The light coming from the screen causes the photocell to respond by generating a pulse. This electric response is transmitted to the processor that identifies the position to which the light pen is pointing. With the movement of light pen over the screen, the lines or images are drawn.

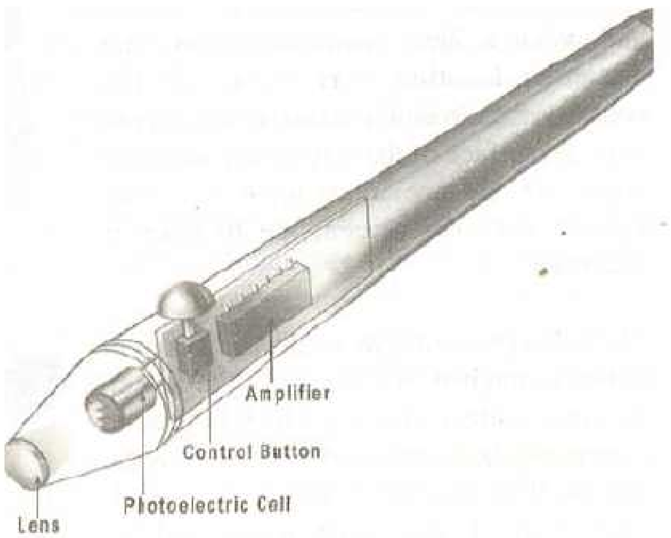


Figure 1-7

It facilitates drawing images and selects objects on the display screen by directly pointing the objects with the pen.

Digital Camera

Digital camera stores images digitally rather than recording them on a film. Once a picture has been taken, it can be downloaded to a computer system and then manipulated with an image editing software and printed. The big advantage of digital cameras is that making photos is both inexpensive and fast because there is no film processing.

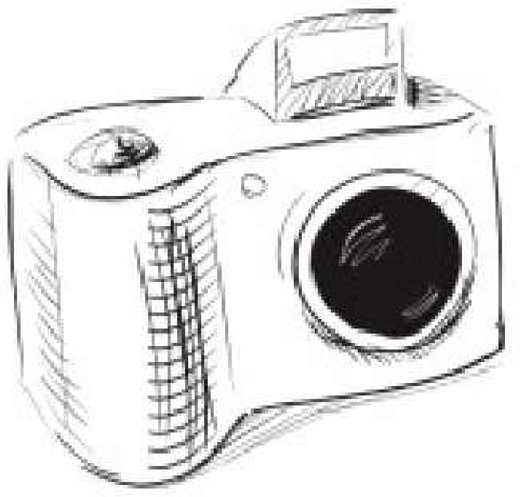


Figure 1-8

All digital cameras record images in an electronic form, that is, the image is represented in computer's language, the language of bits and bytes. Essentially, a digital image is just a long string of 1's and 0's that represent all the tiny colored dots or *pixels* that collectively make up the image. Just like a conventional camera, it has a series of lenses that focus light to create an image of a scene.

Basic difference between digital camera and film-based cameras is that the digital camera does not have a film; it has a sensor that converts light into electrical charges.

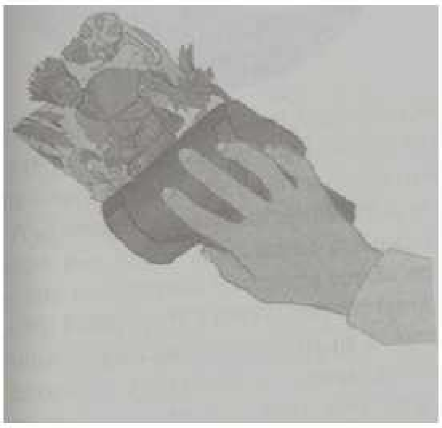
Scanners

There are a number of situations when some information (picture or text) is available on paper and is needed on the computer disk for further manipulation. The simplest way would be to take a photograph of the image directly from the source and convert it into a form that can be saved on the disk. A *scanner* scans an image and transforms the image to ASCII codes (the code used by a computer to represent the characters you find on your keyboard - letters of the alphabet, numbers, punctuation marks, etc.) and graphics. These can be edited, manipulated, combined, and then printed.

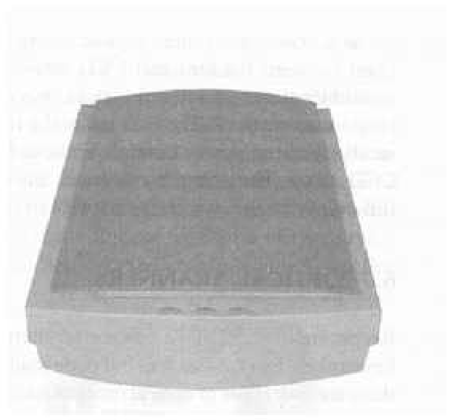
Scanners use a light beam to scan the input data. If the data to be scanned is an image, it can be changed by using the special image editing software. If the image is a page of text, then the special optical character recognition software must be used to convert the images of letters in text and this can be edited by using a word processor.

The two most common types of scanners are *hand-held* scanner and *flat-bed* scanner.

Hand-Held Scanner



(a) **Hand-held scanner**



(b) **Flat-Bed scanner**

Figure 1-9

A hand-held scanner consists of light emitting diodes, which are placed over the material to be scanned. This scanner performs the scanning of the document very slowly from the top to the bottom, with its light on. In this process, all the documents are converted and then stored as an image. While working, the scanner is dragged very steadily and carefully over the document and it should move at a constant speed without stopping, or jerking in order to obtain best results. Due to this reason, hand-held scanners are widely used where high accuracy is not of much importance. The size of the hand-held scanners is small shown in fig 1-9 (a). They come in various resolutions, up to about 800 dpi (dots per inch) and are available in either grey scale or color.

Flat-Bed Scanner

Flat-bed scanners look similar to a photocopier machine. It consists of a box containing a glass plate on its top and a lid that covers the glass plate. This glass plate is primarily used for placing the document to be scanned. The light beam is placed below the glass plate and when it is activated, it moves from left to right horizontally. After scanning one line, the beam of light moves in order to scan the next line and thus, the procedure is repeated until all the lines are scanned. For scanning, an A4 size document takes about 20 seconds. These scanners are capable of scanning black and white as well as color images. The flat-bed scanners are larger in size and more expensive than the hand-held scanners shown in fig. 1-9 (b).

However, they usually produce better quality images because they employ better scanning technology.

Optical Scanners

There are four types of optical recognition: *optical character recognition* (OCR), *optical mark recognition* (OMR), *magnetic ink character recognition* (MICR), and *bar code reader.*

Optical Character Recognition (OCR)

Optical Character Recognition (OCR) is a process of scanning printed pages as images on a flatbed scanner and then using OCR software to recognize the letters as ASCII text. The OCR software has tools for both acquiring the image from a scanner and recognizing the text. In the OCR system, a book or a magazine article is fed directly into an electronic computer file, and then this file is edited by using a word processor. Advanced OCR systems can read text in a large variety of fonts, but they still have difficulty with handwritten text. OCR works best with originals or very clear copies and mono-spaced fonts like Courier.

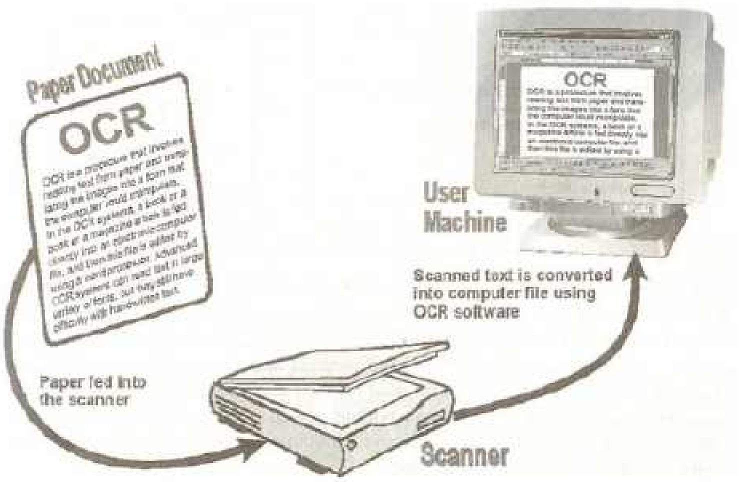

Paper Document
OCR
User Machine
Scanned text is converted into computer file using OCR software
Paper fed into the scanner
Scanner

Figure 1-10

Optical Mark Recognition (OMR)

Optical Mark Recognition (OMR) is the process of detecting the presence of intended marked responses. A mark registers significantly less light than the surrounding paper. Optical mark reading is done by a special device known as *optical mark reader*. The OMR technology enables a high speed reading of large quantities of data and transferring this data to computer without using a keyboard. The OMR reader scans the form, detects the presence of marks, and passes this information to the computer for processing by application software. Generally, this technology is used to read answer sheets (objective type tests). In this method, special printed forms/documents are printed with boxes, which can be marked with dark pencil or ink. These forms are then passed under a light source and the presence of dark ink is transformed into electric pulses, which are transmitted to the computer.

Figure 1-11

Optical mark recognition is also used for standardized testing as well as course enrolment and attendance in education.

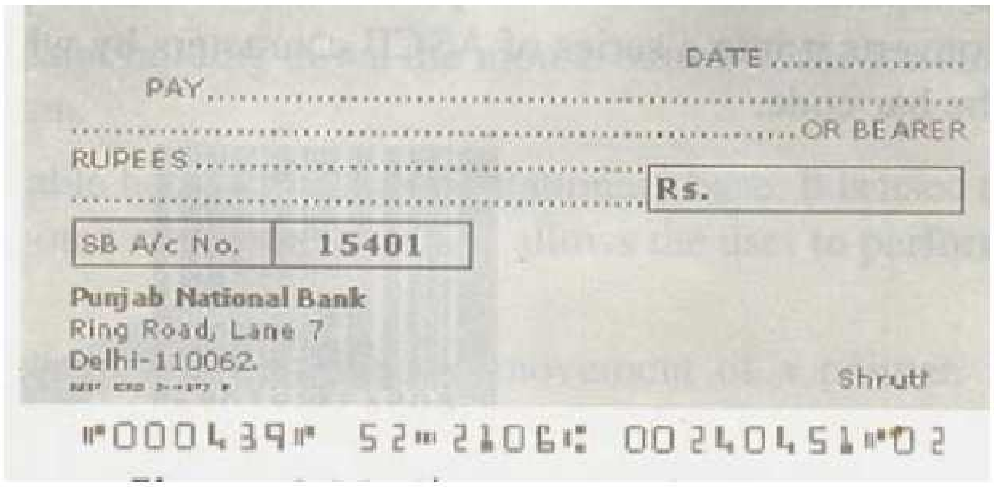
OMR has a better recognition rate than OCR because fewer mistakes are made by machines to read marks than in reading handwritten characters. Large volumes of data can be collected quickly and easily without the need for specially trained staff. Usually, an OMR reader can maintain a throughput of 1500 to 10000 forms per hour. It requires accurate alignment of printing on forms and need a paper of good quality.

Optical mark recognition is traditionally performed using reflective light method where a beam of light is reflected on a sheet with marks, to capture the reflection (presence of mark) or absence of reflection (absence of mark).

Magnetic-Ink Character Recognition (MICR)

Specifically, it refers to the special magnetic encoding, printed on the bottom of a negotiable check. This information is machine readable via bank reader/sorters, which read the visual patterns and magnetic waveforms of the MICR encoding.

The characters are printed using special ink, which contains iron particles that can be magnetized. Magnetic ink character readers are used generally in banks to process the cheque. In case of bank cheque, the numbers written at the bottom are recorded in MICR (using special magnetic ink), representing unique cheque numbers, bank, and branch code, etc. A MICR reads these characters by examining their shapes in a matrix form and the information is then passed on to the computer.



DATE ....

PAY ....

.... OR BEARER

RUPEES .... Rs.

....

SB A/c No. 15401

Punjab National Bank
Ring Road, Lane 7
Delhi-110062.

Shruti

⑈000439⑈ 52⑆2106⑆ 0024045⑈02

Figure 1-12

The banking industry prefers MICR because as compared to the OCR, it gives extra security against forgeries such as color copies of payroll cheque or hand-altered characters on a cheque. The reading speed of the MICR is also higher. This method is very efficient and time saving for data processing.

Bar Code Reader

Bar code is a machine-readable code in the form of a pattern of parallel vertical lines of varying widths. They are commonly used for labeling goods that are available in super markets, numbering books in libraries, etc. These codes/stripes are sensed and read by a photoelectric device (bar code reader) that reads the code by means of reflective light. The information recorded in bar code reader is then fed into the computer, which recognizes the information from the thickness and spacing of bars. Bar code readers are either hand-held or fixed-mount. Hand-held scanners are used to read bar codes on stationary items. With fixed-mount scanners, items having a bar code are passed by the scanner - by hand as in retail scanning applications or by conveyor belt in many industrial applications.

A bar code scanner can record data five to seven times faster than a skilled typist can record. A bar code data entry has an error rate of about I in 3 million. Bar coding also reduces cost in terms of labor and reduced revenue losses resulting from data collection errors.

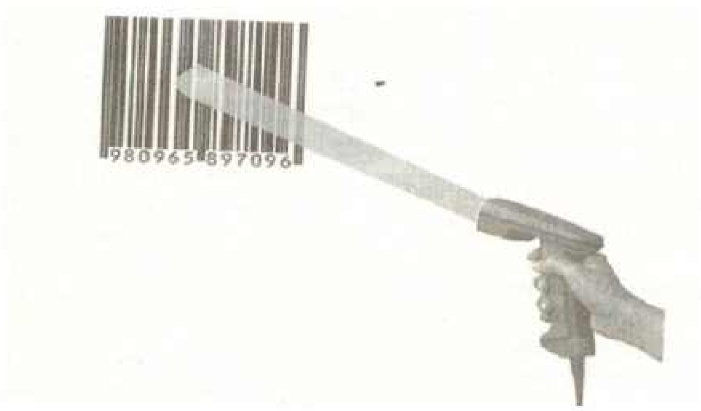


Figure 1-13

Check your progress

Q1. What is the use of input devices in computer?

Q2. Explain

(i) Keyboard
(ii) Mouse
(iii) OMR
(iv) OCR

1.3 Computer Output Units

Output devices convert machine-readable information into human-readable form. The basic functioning of output devices is just the opposite of the input devices, that is, the data is 'fed into' the computer system through the input devices while the output is 'taken out' from the computer through the output devices. However, the output, which comes out from CPU, is in the form of digital signals. The output devices display the processed information by converting them into human-readable form in graphical, alphanumeric or audio-visual forms.

Classification of Output Devices

Output is data that has been processed into a useful form called information. It can be displayed or viewed on a monitor, printed on a printer, or listened through speakers or a headset.

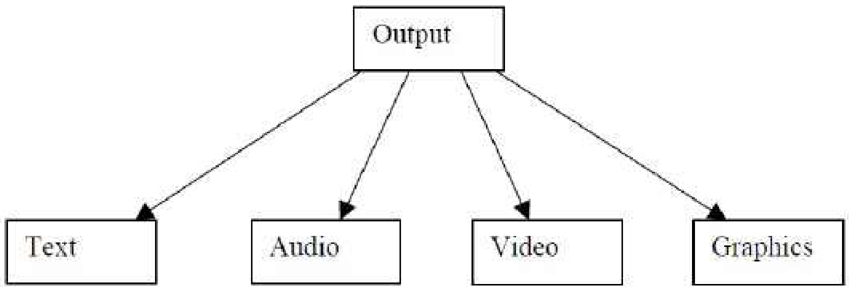


Figure 1-14 Types of Output

There are four basic areas of the output devices:

Text: Textual form of output consists of characters (letters, numbers, punctuation marks, or any other symbol requiring one byte of computer storage space) that are used to create words, sentences, and paragraphs.

Graphics: Graphics are digital representations of non-text information such as drawings, charts, photographs, and animation (a series of still images in sequence that gives the illusion of motion).

Audio: Audio includes music, speech or any sound. A computer converts the sound from a continuous analog signal into a digital format. Most output devices require the computer to convert digital format back into analog signals.

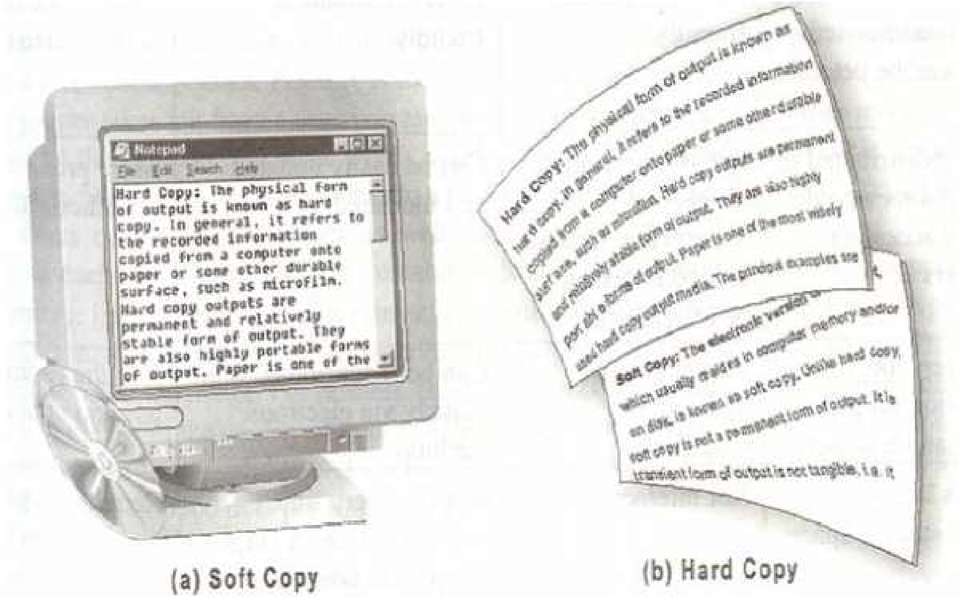
Video: Video consists of images that are played back at speed that provide the illusion of full motion. The images are often captured with a video input device like a video camera. A video capture card is required to convert an analog video signal into a digital signal that the computer can understand. Some output devices accept the digital signal, while others convert the digital signals into analog signals.

The outputs, which can be easily understood and used by human beings, are of following two forms:

1. Hard Copy: The physical form of output is known as *hard copy*. In general, it refers to the recorded information copied from a computer onto paper or some other durable surface, such as microfilm. Hard copy output is permanent and relatively stable form of output. This type of output is also highly portable. Paper is one of the most widely used hard copy output media. The principal examples are printouts, whether text or graphics, from printers. Film, including microfilm and microfiche, is also considered as a hard copy output.

2 Soft Copy: The electronic version of an output, which usually resides in computer memory and or on disk, is known as *soft copy*. Unlike hard copy, soft copy is not a permanent form of output. It is transient and is usually displayed on the screen. This kind of output is not tangible, that is, it cannot be touched. Soft copy output includes audio and visual form of output, which is generated using a computer. In addition, textual or graphical information displayed on a computer monitor is also a soft copy form of output.

Hard copy devices are very slow in operation as compared to the soft copy devices.



```
Hard Copy: The physical form
of output is known as hard
copy. In general, it refers to
the recorded information
copied from a computer onto
paper or some other durable
surface, such as microfilm.
Hard copy outputs are
permanent and relatively
stable form of output. They
are also highly portable forms
of output. Paper is one of the
```

Figure 1-15 two types of outputs

Based on the hard copy and soft copy outputs, the output devices are classified into two types: *hard copy output devices* and *soft copy output devices.*

Hard Copy Output devices

Among the wide variety of the hard copy output devices, *printers,* and *plotters* are the most commonly used. A printer is used to produce printouts of the documents stored on a computer's disk drive. A plotter is a pen-based output device, which is used for producing high quality output by moving ink pens across the paper.

Impact Printers

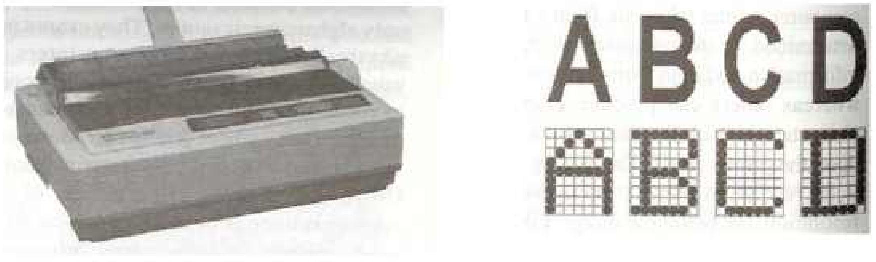
As their names specify, impact printers work by physically striking a head or needle against an ink ribbon to make a mark on the paper. Impact printers are the oldest printing technology and are still in use. An impact printer can print only one character at a time while some impact printers can print an entire line. The three most commonly used impact printers are *dot matrix printers, daisy wheel printers,* and *drum printers.*

Characteristics of impact printers

- In impact printers, there is physical contact with the paper to produce an image.
- They have relatively low consumable costs. The primary recurring costs for these printers are the ink ribbons and paper.
- Due to being robust and low cost, they are useful for bulk printing.
- They can withstand dusty environment, vibrations, and extreme temperature.
- Impact printers are ideal for printing multiple copies (that is, carbon copies) because they can easily print through many layers of paper.
- Due to its striking activity, impact printers are very noisy.
- Since they are mechanical in nature, they tend to be slow.
- Impact printers do not support transparencies.

Dot matrix printers

Dot matrix printer (also known as the *wire matrix printer)* is the oldest printing technology and it prints one character at a time. Usually, dot matrix printers can print any shape of character, which a user can specify. This allows the printer to print many special characters, different sizes of print, and enables it to print graphics, such as charts and graphs. The speed of dot matrix printers is measured in *characters per second* (cps). Most dot matrix printers offer different speeds depending on the quality of print desired. The speed can vary from about 200 to over 500 cps. The print quality is determined by the number of pins (the mechanisms that print the dots), which can vary from 9 to 24. The more pins per inch, the higher the print resolution. The best dot matrix printers (24 pins) can produce near letter quality type image. Most dot matrix printers have a resolution ranging from 72-360 dpi.



(a) Dot Matrix Printer (b) Dot Matrix characters

Figure 1-16 Dot Matrix Printer

Dot matrix printers are inexpensive and have low operating costs. These printers are able to use different types of fonts, different line densities, and different types of paper. Many dot matrix printers are bi-directional, that is, they can print the characters from direction, left or right. The major limitation of dot matrix printer is that it prints only in black and white. The image printing ability is also very limited. These printers may not be able to print graphic objects adequately but can handle applications such as accounting, personnel, and payroll very well. Dot matrix printers are commonly used in low-cost, low-quality applications like cash registers. These printers are limited to situations where carbon copies are needed and the quality is not too important.

Working of a dot matrix printer

The technology behind dot matrix printing is quite simple. The paper is pressed against a drum (a rubber-coated cylinder) and is intermittently pulled forward as printing progresses. The printer consists of an electro-magnetically driven print head, which is made up of numerous print wires (pins). The characters are formed by moving the electro-magnetically driven print head across the paper, which strikes the printer ribbon situated between the paper and print head pin. As the head stamps onto the paper through the inked ribbon, a character is produced that is made up of these dots. These dots seem to be very small for the normal vision and appear like solid human readable characters.

Daisy wheel printers

It is named so because the print head of this printer resembles a daisy flower, with the printing arms that appear like the petals of the flower. These printers are commonly referred to as *letter quality printers* as the print quality is as good as that of a high-quality typewriter.

Daisy wheel printers produce high-resolution output and are more reliable than dot matrix printers. They can have speeds up to 90 cps. These printers are also called as *smart printers* because of its bidirectional printing and built-in microprocessor control features.

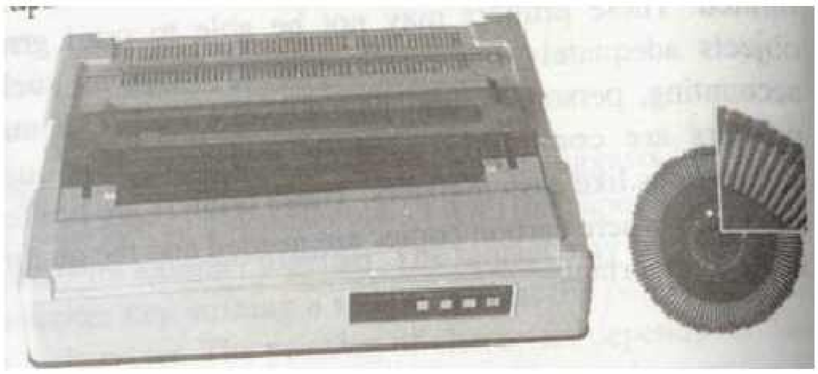


Figure 1-17 Daisy Wheel Printer

However, daisy wheel printers give only alphanumeric output. They cannot print graphics and cannot change fonts unless the print wheel is physically replaced: These printers are usually very slow because of the time required to rotate the print wheel for each character desired. Daisy wheel printers are slower and more expensive than dot matrix printers.

Working of a daisy wheel printer

These printers have print heads composed of metallic or plastic wheels. A raised character is placed on the tip of each of the daisy wheels 'petals'. Each petal has an appearance of a letter (upper case and lower case), number or punctuation mark on it. To print, the print wheel is rotated around until the desired character is under the print hammer. The petal is then struck from behind by the print hammer, which strikes the character, pushing it against the ink ribbon, and onto the paper, creating the character.

Drum printers

Such types of printers print an entire line in a single operation. Such printers are known as *line printers*. Drum printer is one of the most commonly used line printers. This arrangement allows a continuous high-speed printing. Its printing speed varies from 150 lines to 2500 lines per minute with 96 to 160 characters on a 15-inch line. Although, such printers are much faster than character printers, they tend to be quite loud, have limited multi-font capability, and often produce lower print quality than most recent printing technologies. Line printers are designed for heavy printing applications. For example, in businesses where enormous amounts of materials are printed.

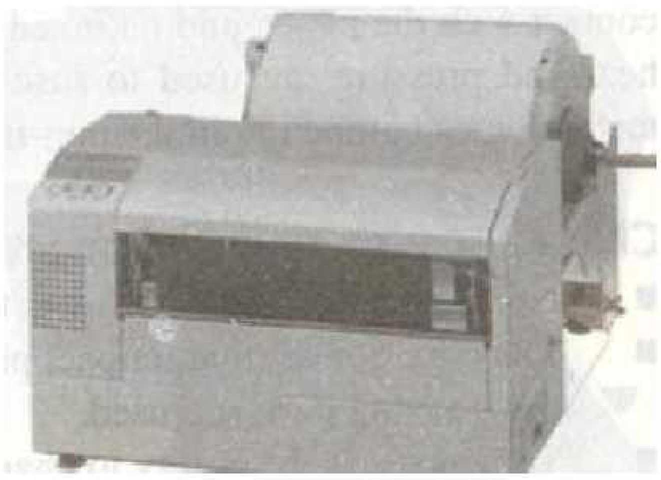


Figure 1-18: Drum Printer

Working of a drum printer

The basics of a line printer like drum printer are similar to those of a serial impact printer forms characters and images printers. There is no striking multiple type elements against the paper almost simultaneously, so that an entire line is printed in one operation. A typical arrangement of a drum printer involves a large rotating drum mounted horizontally and positioned in front of a very wide, inked ribbon, which in turn is positioned in front of the paper itself. The drum contains characters molded onto its surface in columns around its circumference; each column contains a complete set of characters (letters, digits, etc.) running around the circumference of the drum. The drum spins continuously at high speed when the printer is operating. In order to print a line, hammers positioned behind the paper ram the paper against the ribbon and against the drum beyond it at exactly the right instant; such that the appropriate character is printed in each column as it spins past on the drum. Once every column has been printed, the paper is advanced upward so that the next line can be printed.

Non-Impact Printers

Unlike impact printers, a non-impact printer forms characters and images without making direct physical contact between printing mechanism and paper. In this printer, the print head does not make contact with the paper, and no inked ribbon is required. Ink can be sprayed against the paper and then heat and pressure are used to fuse a fine black powder into the shape of a character. They use techniques other than physically striking the page to transfer ink onto the page. The major technologies competing in the non-impact market are *ink-jet* and *laser*.

Characteristics of non-impact printers

Non-impact printers are faster than impact

- They are quieter than impact printers because there is no striking mechanism involved and only few moving parts are used. They possess the ability to change typefaces automatically.
- These printers produce high-quality graphics.

- These printers usually support the transparencies.
- These printers cannot print multiple forms because no impact is being made on the paper.

Ink-Jet Printer

It is the most type of printer used in home. Being a non-impact it does not touches the paper while creating an image. It uses a series of nozzles to spray print multiple onto the paper. Originally it was made black and white only. However, the print head has now been expanded and the nozzle accommodates CMYK. The combination of these four colors will be the resultant color.

These printers are costlier than the dot-matrix printer, but the quality is much better. Ink-jet printers typically print with a resolution of 600 dpi or more. Due to the high resolution, these printers produce high quality graphics and text printouts. They are also affordable, which appeals to small businesses and home offices. These printers print documents at a medium pace, but slow down if printing a document with multicolor. These printers can print about 6 pages a minute. Moreover, they can also be programmed to print unusual symbols such as Japanese or Chinese characters.

Working of an ink-jet printer

An ink-jet printer has a print cartridge with a series of tiny electrically heated chambers. These cartridges are attached to print heads with a series of small nozzles that spray ink onto the surface of the paper. As print head moves back and forth across the page, software gives instructions regarding the type and the quantity of colors. It also tells the position where the dots of ink should be 'sprayed'. There are two main ways to drop the ink droplets, namely, the *bubble-jet* and *piezo-electric* technology.

Figure 1-19: Ink-jet Printer

Bubble-jet printers use heat to fire ink onto the paper. Piezo-electric technology uses a piezo crystal at the back of the ink reservoir.

Laser printers

A laser printer provides the highest quality text and images for personal computers today, operates on the same principle as that of a photocopy machine. They are also known as *page printers* because they process and store the entire page before they actually print it.

Characteristics of Laser printer

- It is a very fast printer.
- It can print text and graphics with a very high quality resolution from 300 to 1200 dpi.
- It can print in different fonts, that is, type styles and sizes.
- It is more expensive to buy and maintain than the other printers.

Figure 1-20: Laser Printer

Working of a laser printer

The core component of laser printing system is the photoreceptor drum. A rotating mirror inside the printer causes the beam of a laser to sweep across the photoconductive drum. Initially, the beam of laser charges the photoconductive drum positively. When the charged photoconductor is exposed to an optical image through a beam of light to discharge, a latent or invisible image is formed. At the point where the laser strikes the surface of drum, it creates a dot of positive charge. These points are represented by a black dot, which will be printed on the paper. After this, the printer coats the drum with a container, which contains a black powder called *toner*. This toner is negatively charged, and so it clings to the positive areas of the drum surface. When the powder pattern gets fixed, the drum is rotated and the paper is fed into the drum surface via a pressure roller. This pressure roller transfers the black toner onto the paper. Since the paper is moving at the same speed as the drum, the paper picks up the image pattern precisely. Finally, the printer passes the paper through the fuser, a pair of heated rollers. As the paper passes through these rollers, the loose toner powder gets melted and fuses with the fibers in the paper.

Plotters

A plotter is a pen-based output device that is attached to a computer for making vector graphics, that is, images created by a series of many straight lines. It is used to draw high-resolution charts, graphs, blueprints, maps, circuit diagrams, and other line-based diagrams. Plotters are similar to printers, but they draw lines using a pen. As a result, they can produce continuous lines, whereas printers can only simulate lines by printing a closely spaced series of dots. Multicolor plotters use different-colored pens to draw different colors. Color plots can be made by using four pens (cyan, magenta, yellow, and black) and need no human intervention to change them.

Plotters are relatively expensive as compared to printers but can produce more printouts than standard printers. They are mainly used for Computer Aided Design (CAD) and Computer Aided Manufacturing (CAM) applications such as printing out plans for houses or car parts. These are also used with programs like AUTO CAD (computer assisted drafting) to give graphic outputs.

Types of Plotters

There are two different types of plotters, one where the paper moves (drum), and the other where the paper is stationary (flatbed plotter).

***Drum Plotters*:** In drum plotters, the paper on which the design is to be made is placed over a drum. These plotters consist of one or more pen(s) that are mounted on a carriage and this carriage is horizontally placed across the drum. The drum can rotate in either clockwise or anticlockwise direction under the control of plotting instructions sent by the computer. Drum plotters are used to produce continuous output, such as plotting earthquake activity, or for long graphic output, such as tall building structures.

***Flatbed Plotters*:** Flatbed plotters consist of a stationary horizontal plotting surface on which paper is fixed. The pen is mounted on a carriage, which can move horizontally, vertically, leftwards or rightwards to draw line. In flatbed plotters, the paper does not move, the pen-holding mechanism provides all the motion. These plotters are instructed by the computer on the movement of pens in the X- Y coordinates on the page. These plotters are capable of working on any standard, that is, from A4 size paper to some very big beds. Depending on the size of the flatbed surface, these are used in designing of ships, aircrafts, buildings, etc. The major disadvantage of this plotter is that it is a slow output device and can take hours to complete a complex drawing.

(a) Drum Plotter

(b) Flatbed Plotter

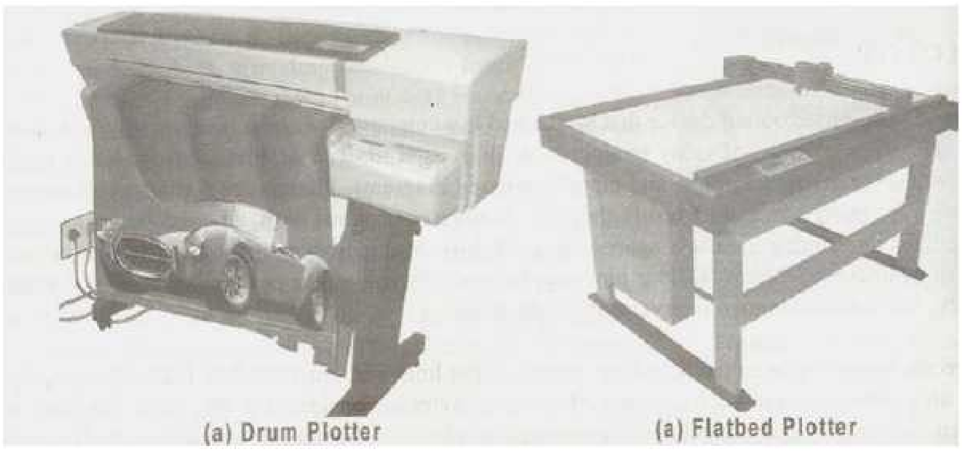


Figure 1-20

Check your progress

Q1. What do you mean by output devices? Why a computer need output device?

Q2. Differentiate between impact and non-impact computer.

1.4 Summary

In this unit you learnt about the input and output devices of the computer. This unit also tells about the detail information about the input units (such as mouse, keyboard, scanner, etc.) and output unit (such as monitor, printer, plotters, etc.)

- Input devices are used to send information to CPU.
- Input devices are directly connected to CPU.
- Most Commonly input devices are Mouse, Keyboard, Joystick, Scanner, etc.
- Output devices are mainly used to get information from CPU.
- Output devices are also directly connected to CPU.

1.5 Review Questions

Q1. What are Input and Output devices? Explain their working.

Q2. What is the difference between mouse and trackball? Explain with example.

Q3. How many types of printer available in the market? Explain its working of every Printer.

Q4. What do you mean by Plotter? Write some similarities and differences between printer and plotters.

Q5. What is projector? Is it a output device? Explain the working of it.

UNIT-II

Computer Memory

Structure

2.0 Introduction

2.1 Memory –Introduction

2.2 Memory Cell

2.3 Memory Organization

2.4 Read Only Memory

2.5 Serial Access Memory

2.6 Summary

2.7 Review Questions

2.0 Introduction

In this unit we focused on computer memory. In this unit there are seven sections. In Sec. 2.1 you will learn about memory. As you know about memory it is any physical device capable of storing information temporarily or permanently. For example, Random Access Memory (RAM), is a volatile memory that stores information on an integrated circuit used by the operating system, software, and hardware. In Sec. 2.2 you will learn about Memory Cell. The memory cell is the fundamental building block of computer memory. The memory cell is an electronic circuit that stores one bit of binary information and it must be set to store a logic 1 (high voltage level) and reset to store a logic 0 (low voltage level). Its value is maintained/stored until it is changed by the set/reset process. The value in the memory cell can be accessed by reading it. In Sec. 2.3 you will know about memory organization. In Sec. 2.4 we focused on Read Only Memory and in the Sec. 2.5we introduced Serial Access Memory. In Sec. 2.6 and 2.7 you will find summary and review questions respectively.

Objectives

After studying this unit you should be able to:

- Express memory, memory cells.
- Define Memory organization
- Describe Read Only Memory & Serial Access Memory

2.1 Memory -Introduction

A memory is just like a human brain. It is used to store data and instruction. Computer memory is the storage space in computer where data is to be processed and instructions required for processing are stored.

The memory is divided into large number of small parts. Each part is called cell. Each location or cell has a unique address which varies from zero to memory size minus one.

For example if computer has 64k words, then this memory unit has 64 * 1024=65536 memory location. The address of these locations varies from 0 to 65535.

2.2 Memory Cells

Primary computer memory is made up of memory cells, where each memory cell contains exactly one number. Thus, a memory cell can be thought of as box into which a single number can be placed. The number contained in a memory cell can be changed over time. When a new number is stored into a memory cell, the old number contained in the memory cell is lost forever. At any time, the computer may peer into a memory cell to read the current contents of the memory cell. The computer may read the contents of a memory cell as many times as it wants, without disturbing it.

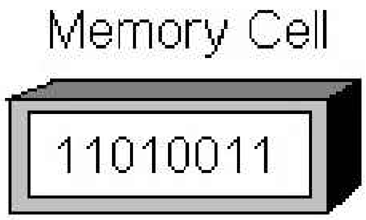


Figure 2-1

A typical computer has millions of memory cells, and every memory cell has a name. While a small number of memory cells are given specialized names, such as `Program Counter' or `Processor Status Register', most memory cells are named using a unique number. (Remember, from the previous unit, *everything* in a computer is a number!) The unique number that names a memory cell is called its `address.' No two memory cells have the same address. The address of a memory cell never changes over time. Going back to the box analogy of the previous paragraph, the address of a memory cell can be thought of as a number permanently imprinted on the side of the box in indelible ink. Thus, every memory cell has two numbers -- its address (which never changes) and its current contents (which changes over time.)

Since a memory cell has both a contents and an address, both of which are numbers, it is useful to adopt a notation to keep them separate. A fairly common notation is to enclose the memory cell address in square brackets, followed by a colon, followed by the memory cell contents. For example, `[223]: 17' is the notation used to specify that memory cell [223] currently has the number 17 as its contents.

As an example, consider the two arbitrary memory cells named [223] and [873]. When power is first applied to these two memory cells, both cells will have an initial content of zero; this is shown below:

[223]: 0 [873]: 0

Eventually, the computer comes along and writes the number 104 into memory cell [223]; this is shown below:

[223]: 104 [873]: 0

Later on, the computer writes the number 105 into memory cell [873]; this is shown below:

[223]: 104 [873]: 105

Even later on, the computer overwrites memory cell [223] with 72. The previous contents of memory cell [223] are lost forever; this is shown below:

[223]: 72   [873]: 105

Finally, at some even later point in time, the number 104 is stored back into memory cell [223]. Again, the previous contents of memory cell [223] are lost forever; this is shown below:

[223]: 104   [873]: 105

The computer reads and overwrites the contents of memory cells [223] and [873] as many times as needed to accomplish its current task.

Memory Cell Address

Since a memory consists of a large number of memory cells, there has to be a way to keep track of which cell is which.

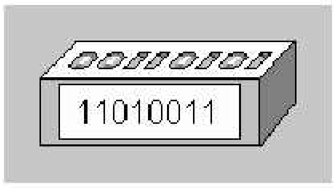


Figure 2-2

Thus each cell in a memory has an index number called its *address* which is sort of "chiseled" into the "edge" of the cell.

The cell pictured here has a content of **11010011_2 = $D3_{16}$** and an address of **00110101_2 = 35_{16}**
A cell address CANNOT be changed

The address numbers *start at 0* and go 1, 2, 3, etc.

The address of a cell is usually different than its content - although they *can* be the same.

2.3 Memory Organization

Computers employ many different types of memory (semi-conductor, magnetic disks, and USB sticks, DVDs etc.) to hold data and programs. Each type has its own characteristics and uses. We will look at the way that Main Memory (RAM) is organized and briefly at the characteristics of Register Memory and Disk Memory. Let us locate these 3 types of memory in a simplified model of a computer:

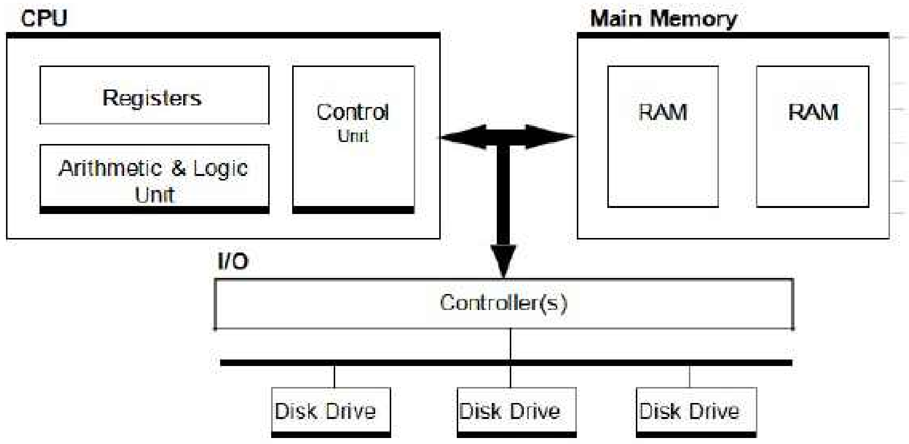


Figure 2-3

Register Memory

Registers are memories located within the Central Processing Unit (CPU). They are few in number (there are rarely more than 64 registers) and also small in size, typically a register is less than 64 bits; 32-bit and more recently 64-bit are common in desktops.

The contents of a register can be "read" or "written" very quickly[1] however, often an order of magnitude faster than main memory and several orders of magnitude faster than disk memory.

Different kinds of register are found within the CPU. General Purpose Registers[2] are available for general[3] use by the programmer. Unless the context implies otherwise we will use the term "register" to refer to a General Purpose Register within the CPU. Most modern CPU"s have between 16 and 64 general purpose registers. Special Purpose Registers have specific uses and are either non-programmable and internal to the CPU or accessed with special instructions by the programmer. Examples of such registers that we will encounter later in the course include: the Program Counter register (PC), the Instruction Register (IR), the ALU Input & Output registers, the Condition Code (Status/Flags) register, the Stack Pointer register (SP). The size (the number of bits in the register) of these registers varies according to register type. The Word Size of an architecture is often (but not always!) defined by the size of the general purpose registers.

In contrast to main memory and disk memory, registers are referenced directly by specific instructions or by encoding a register number within a computer instruction. At the programming (assembly) language level of the CPU, registers are normally specified with special identifiers (e.g. R0, R1, R7, SP, PC).

As a final point, the contents of a register are lost if power to the CPU is turned off, so registers are unsuitable for holding long-term information or information that is needed for retention after a power-shutdown or failure. Registers are however, the fastest memories, and if exploited can result in programs that execute very quickly.

Main Memory (RAM)

If we were to sum all the bits of all registers within CPU, the total amount of memory probably would not exceed 5,000 bits. Most computational tasks undertaken by a computer require a lot more memory. Main memory is the next[4] fastest memory within a computer and is much larger in size. Typical main memory capacities for different kinds of computers are: PC 512MB[5], fileserver 4GB, database server 8GB. Computer architectures also impose an architectural constraint on the maximum allowable RAM. This constraint is normally equal to $2^{WordSize}$ memory locations.

RAM[6] (Random[7] Access Memory) is the most common form of Main Memory. RAM is normally located on the motherboard and so is typically less than 12 inches from the CPU. ROM (Read Only Memory) is like RAM except that its contents cannot be overwritten and its contents are not lost if power is turned off (ROM is non-volatile).

Although slower than register memory, the contents of any location[8] in RAM can still be "read" or "written" very quickly[9]. The time to read or write is referred to as the **access time** and is constant for all RAM locations.

In contrast to register memory, RAM is used to hold both program code (instructions) and data (Numbers strings etc.). Programs are "loaded" into RAM from a disk prior to execution by the CPU.

Locations in RAM are identified by an **addressing scheme***e.g.* numbering the bytes in RAM from 0 onwards[10]. Like registers, the contents of RAM are lost if the power is turned off.

Disk Memory

Disk memory[11] is used to hold programs and data over the longer term. The **contents of a disk are NOT lost if the power is turned off.** Typical hard disk capacities range from 100GB to over 1TB ($1x10^{30}$). Disks are much slower than register and main memory, the access-time (known as the seek-time) to data on disk is typically between 2 and 4 milli-seconds, although disk drives can transfer thousands of bytes in one go achieving transfer rates from 25MB/s to 500MB/s.

Disks can be housed internally within a computer "box" or externally in an enclosure connected by a fast USB or firewire cable[12]. Disk locations are identified by special disk addressing schemes (e.g. track and sector numbers).

Summary of Characteristics

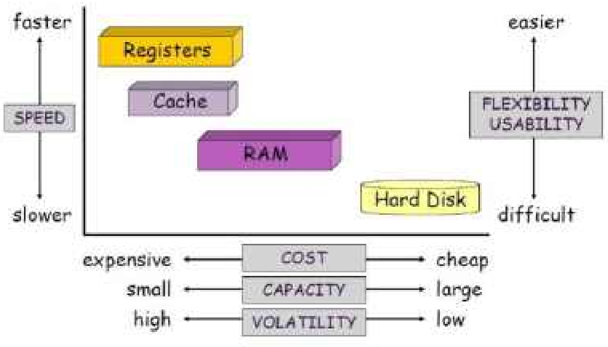


Figure 2-4

SRAM, DRAM, SDRAM, DDR SDRAM

There are many kinds of RAM and new ones are invented all the time. One aim is to make RAM access as fast as possible in order to keep up with the increasing speed of CPUs.

SRAM (Static RAM) is the fastest form of RAM but also the most expensive. Due to its cost it is not used as main memory but rather for cache memory. Each bit requires a 6-transistor circuit.

DRAM (Dynamic RAM) is not as fast as SRAM but is cheaper and is used for main memory. Each bit uses a single capacitor and single transistor circuit. Since capacitors lose their charge, DRAM needs to be refreshed every few milliseconds. The memory system does this transparently. There are many implementations of DRAM, two well-known ones are SDRAM and DDR SDRAM.

SDRAM (Synchronous DRAM) is a form of DRAM that is synchronized with the clock of the CPU"s system bus, sometimes called the front-side bus (FSB). As an example, if the system bus operates at 167Mhz over an 8-byte (64-bit) data bus , then an SDRAM module could transfer 167 x 8 ~ 1.3GB/sec.

DDR SDRAM (Double-Data Rate DRAM) is an optimization of SDRAM that allows data to be transferred on both the rising edge and falling edge of a clock signal. Effectively doubling the amount of data that can be transferred in a period of time, For example a PC-3200 DDR-SDRAM module operating at 200Mhz can transfer 200 x 8 x 2 ~ 3.2GB/sec over an 8-byte (64-bit) data bus.

ROM, PROM, EPROM, EEPROM, Flash

In addition to RAM, there are also a range of other semi-conductor memories that retain their contents when the power supply is switched off.

ROM (Read Only Memory) is a form of semi-conductor that can be written to once, typically in bulk at a factory. ROM was used to store the "boot" or start-up program (so called firmware) that a computer executes when powered on, although it has now fallen out-of-favour to more flexible memories that support occasional writes. ROM is still used in systems with fixed functionalities, e.g. controllers in cars, household appliances etc.

PROM (Programmable ROM) is like ROM but allows end-users to write their own programs and data. It requires special PROM writing equipment. Note: users can only write-once to PROM.

EPROM (Erasable PROM): - With EPROM we can erase (using strong ultra-violet light) the contents of the chip and rewrite it with new contents, typically several thousand times. It is commonly used to store the "boot" program of a computer, known as the firmware. PCs call this firmware, the BIOS (Basic I/O System). Other systems use Open Firmware. Intel-based Macs use EFI (Extensible Firmware Interface).

EEPROM (Electrically Erasable PROM): - As the name implies the contents of EEPROMs are erased electrically. EEPROMSs are also limited to the number of erase-writes that can be performed (e.g, 100,000) but support updates (erase-writes) to individual bytes whereas EPROM updates the whole memory and only supports around 10,000 erase-write cycles.

FLASH memory is a cheaper form of EEPROM where updates (erase-writes) can only be performed on blocks of memory, not on individual bytes. Flash memories are found in USB sticks, flash cards and typically range in size from 1GB to 32GB. The number of erase/write cycles to a block is typically several hundred thousand before the block can no longer be written.

Main Memory Organization

Main memory can be considered to be organized as a matrix of bits. Each row represents a memory location, the number of bits in which is often the word size of the architecture, although it can be a word multiple (e.g. two words) or a partial word (e.g. half word). **For simplicity we will assumethat data within main memory can only be read or written a single row (memory location) at a time.** For a 96-bit memory we could organize the memory as 12x8 bits, or 8x12 bits or, 6x16 bits, oreven as 96x1 bits or 1x96 bits. Each row also has a natural number called its **address** which is used for selecting the row:

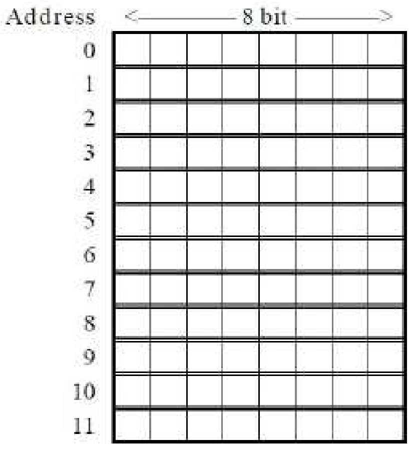


Figure 2-5

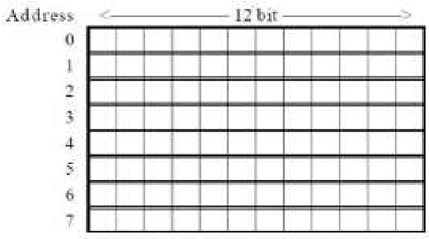


Figure 2-6

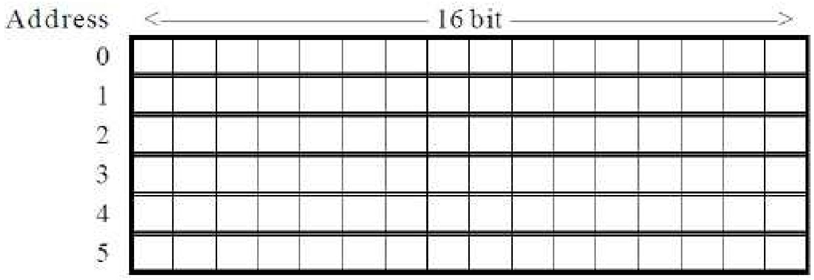


Figure 2-7

Byte Addressing

Main-memories generally store and recall rows, which are multi-byte in length (e.g. 16-bit word = 2 bytes, 32-bit word = 4 bytes). Many architectures, however, make main memory **byte-addressable** rather than **word addressable.** In such architectures the CPU and/or the main memory hardware is capable of reading/writing any individual byte. Here is an example of a main memory with 16-bit memory locations[13]. Note how the memory locations (rows) have even addresses.

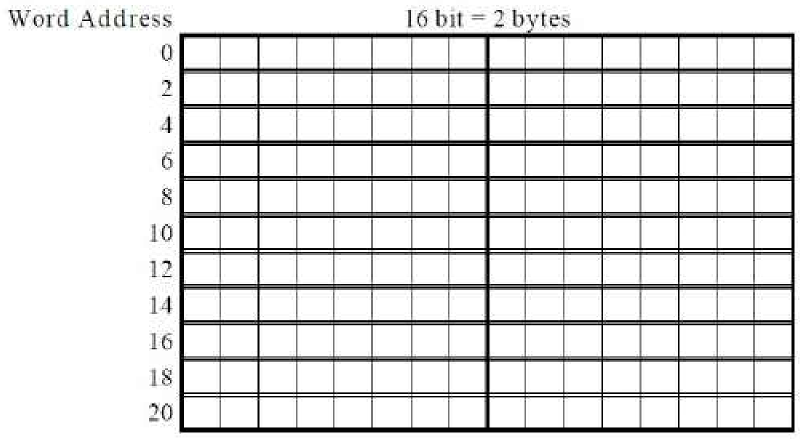


Figure 2-8

Byte Ordering

The ordering of bytes within a **multi-byte** data item defines the endian-ness of the architecture.

In BIG-ENDIAN systems the most significant byte of a multi-byte data item always has the lowest address, while the least significant byte has the highest address.

In LITTLE-ENDIAN systems, the least significant byte of a multi-byte data item always has the lowest address, while the most significant byte has the highest address.

In the following example, table cells represent bytes, and the cell numbers indicate the address of that byte in main memory. Note: by convention we draw the bytes within a memory word left-to-right for big-endian systems, and right-to-left for little-endian systems.

Word Address	**Big-Endian**			
0	0	1	2	3
4	4	5	6	7
8	8	9	10	11
12	12	13	14	15

Word Address	**Little-Endian**			
0	3	2	1	0
4	7	6	5	4
8	11	10	9	8
12	15	14	13	12

Figure 2-9

MSB ————> LSB MSB ————> LSB

Note: an N-character ASCII string value is not treated as one large multi-byte value, but rather as N byte values, i.e. the first character of the string always has the lowest address, the last character has the highest address. This is true for both big-endian and little-endian. An N-character Unicode string would be treated as N two-byte values and each two-byte value would require suitable byte-ordering.

Example: Show the contents of memory at word address 24 if that word holds the number given by122E 5F01H in both the big-endian and the little-endian schemes?

Big Endian

	MSB	——>		LSB
24		25	26	27
Word 24	12	2E	5F	01

Little Endian

	MSB	——>		LSB
27		26	25	24
Word 24	12	2E	5F	01

Example: Show the contents of main memory from word address 24 if those words hold the ASCIIstring value JIM SMITH.

Big Endian

	+0	+1	+2	+3
Word 24	J	I	M	
Word 28	S	M	I	T
Word 32	H	?	?	?

Little Endian

	+3	+2	+1	+0
Word 24		M	I	J
Word 28	T	I	M	S
Word 32	?	?	?	H

The bytes labeled with ? are unknown. They could hold important data, or they could be don"t care bytes – the interpretation is left up to the programmer.

Unfortunately computer systems[14], in use today are split between those that are big-endian, and those that are little-endian[15]. This leads to problems when a big-endian computer wants to transfer data to a little-endian computer. Some architectures, for example the PowerPC and ARM, allow the endian-ness of the architecture to be changed programmatically.

Word Alignment

Although main-memories are generally organized as byte-addressed rows of words and accessed a row at a time, some architectures, allow the CPU to access any word-sized bit-group regardless of its byte address. We say that accesses that begin on a memory word boundary are **aligned accesses** while accesses that do not begin on word boundaries are **unaligned accesses**.

Address	Memory (16-bit) word		
0	**MSB**	**LSB**	Word starting at Address 0 is Aligned
2			
4		**MSB**	Word starting at Address 5 is Unaligned
6	**LSB**		

Reading an unaligned word from RAM requires (i) reading of adjacent words, (ii) selecting the required bytes from each word and (iii) concatenating those bytes together => SLOW. Writing an unaligned word is more complex and slower[16]. For this reason some architectures prohibit unaligned word accesses. e.g. on the 68000 architecture, words must not be accessed starting from an odd-address (e.g. 1, 3, 5, 7 etc), on the SPARC architecture, 64-bit data items must have a byte address that is a multiple of 8.

Memory Modules, Memory Chips

So far, we have looked at the logical organization of main memory. Physically RAM comes on small memory modules (little green printed circuit-boards about the size of a finger). A typical memory module holds 512MB to 2GB. The computer's motherboard will have slots to hold 2, 4 maybe 8 memory modules. Each memory module is itself comprised of several memory chips. For example here are 3 ways of forming a 256x8 bit memory module.

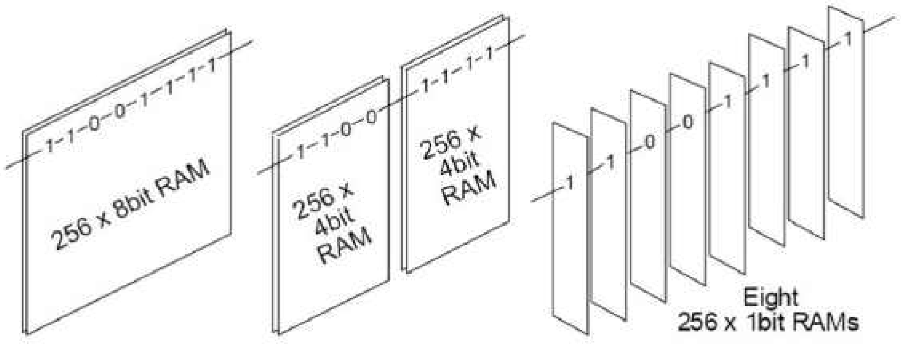


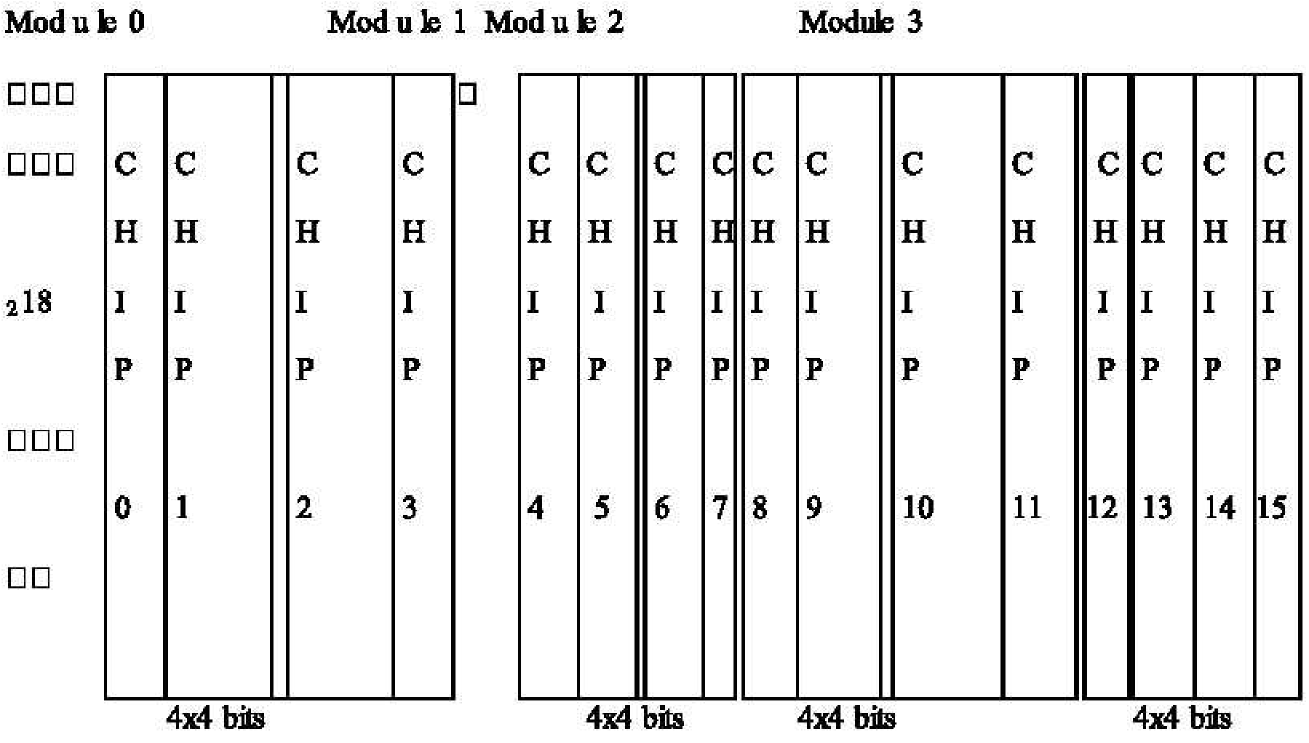
Figure 2-10

In the first case, main memory is built with a single memory chip. In the second, we use two memory chips; one gives us the most significant 4 bits, the other, the least significant 4 bits. In the third we use 8 memory chips, each chip gives us 1 bit - to read an 8 bit memory word, we would have to access all 8 memory chips simultaneously and concatenate the bits.

On PCs, memory modules are known as DIMMs (dual inline memory modules) and support 64-bit transfers. The previously generation of modules were called SIMMs (single inline memory modules) and supported 32-bit data transfers.

Example: Given Main Memory = 1M x 16 bit (word addressable),

RAM chips = 256K x 4 bit



RAM chips per memory module = $\frac{\text{Width of Memory Word}}{\text{Width of RAM Chip}} = 16/4 = 4$

18 bits are required to address a RAM chip (since 256K = 2^{18} = Length of RAM Chip)

A 1Mx16 bit word-addressed memory requires 20 address bits (since 1M =2^{20})

Therefore 2 bits (=20–18) are needed to select a module.

The total number of RAM Chips = (1M x 16) / (256K x 4) = 16.

Total number of Modules = Total number of RAM chips / RamChipsPerModule = 16/4 = 4

Interleaved Memory

When memory consists of several memory chips, some address bits will select the chip, and the remaining bits will select a row within the selected chip. When the module selection bits are the least significant bits of the memory address, we call the resulting memory a **low-order interleaved** memory. When the module selection bits are the most significant bits of the memory address, we call the resulting memory a **high-order interleaved** memory. Interleaved memory can yield performance advantages if more than one memory chip can be read/written at a time:-

(i) For low-order interleave if we can read the same row in each chip. This is good for a single multi-word access of sequential data such as program instructions, or elements in a vector,

(ii) For high-order interleave, if different chips can be independently accessed by different units. This is good if the CPU can access rows in one chip, while at the same time, the hard disk (or a second CPU) can access different rows in another chip concurrently.

Example: Given that Main Memory = 1Mx8bits, RAM chips = 256K x 4bit. For this memory we would require 4x2=8 RAM chips. Each chip would require 18 address bits (ie. 2^{18} = 256K) and the full 1Mx16 bit memory would require 20 address bits (ie. 2^{20} = 1M).

Check your progress

Q1. Define memory and memory cell.

Q2. How a computer organize the memory?

Q3. Define SRAM, DRAM, SDRAM, DDR SDRAM

2.4 Read-only memory

Read-only memory (ROM) is a class of storage medium used in computers and other electronic devices. Data stored in ROM cannot be modified, or can be modified only slowly or with difficulty, so it is mainly used to distribute firmware (software that is very closely tied to specific hardware and unlikely to need frequent updates).

In its strictest sense, ***ROM*** refers only to mask ROM (the oldest type of solid state ROM), which is fabricated with the desired data permanently stored in it, and thus can never be modified. Despite the simplicity, speed and economies of scale of mask ROM, field-programmability often make reprogrammable memories more flexible and inexpensive. As of 2007, actual ROM circuitry is therefore mainly used for applications such as microcode, and similar structures, on various kinds of processors.

Other types of non-volatile memory such as erasable programmable read only memory (EPROM) and electrically erasable programmable read-only memory (EEPROM or Flash ROM) are sometimes referred to, in an abbreviated way, as "read-only memory" (ROM); although these types of memory can be erased and re-programmed multiple times, writing to this memory takes longer and may require different procedures than reading the memory.[1] When used in this less precise way, "ROM" indicates a *non-volatile* memory which serves functions typically provided by mask ROM, such as storage of program code and nonvolatile data.

Figure 2-11

There are two main reasons that read-only memory is used for certain functions within the PC:

- **Permanence:** The values stored in ROM are always there, whether the power is on or not. A ROM can be removed from the PC, stored for an indefinite period of time, and then replaced, and the data it contains will still be there. For this reason, it is called *non-volatile storage*. A hard disk is also non-volatile, for the same reason, but regular RAM is not.
- **Security:** The fact that ROM cannot easily be modified provides a measure of security against accidental (or malicious) changes to its contents. You are not going to find viruses infecting true ROMs, for example; it's just not possible. (It's technically possible with *erasable* EPROMs, though in practice never seen.)

Read-only memory is most commonly used to store system-level programs that we want to have available to the PC at all times. The most common example is the system BIOS program, which is stored in a ROM called (amazingly enough) the *system BIOS ROM*. Having this in a permanent ROM means it is available when the power is turned on so that the PC can use it to boot up the system. Remember that when you first turn on the PC the system memory is empty, so there has to be *something* for the PC to use when it starts up.

There are four basic ROM types:

1. PROM - Programmable Read Only Memory
2. EPROM - Erasable Programmable Read Only Memory
3. EEPROM - Electrically Erasable Programmable Read Only Memory
4. Flash EEPROM memory

PROM

Creating ROM chips totally from scratch is time-consuming and very expensive in small quantities. For this reason, developers created a type of ROM known as programmable read-only memory (PROM). Blank PROM chips can be bought inexpensively and coded by the user with a programmer.

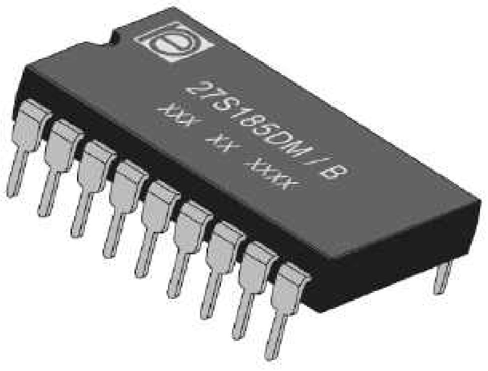


Figure 2-12

PROM chips have a grid of columns and rows just as ordinary ROMs do. The difference is that every intersection of a column and row in a PROM chip has a fuse connecting them. A charge sent through a column will pass through the fuse in a cell to a grounded row indicating a value of 1. Since all the cells have a fuse, the initial (blank) state of a PROM chip is all 1s. To change the value of a cell to 0, you use a programmer to send a specific amount of current to the cell. The higher voltage breaks the connection between the column and row by burning out the fuse. This process is known as burning the PROM.

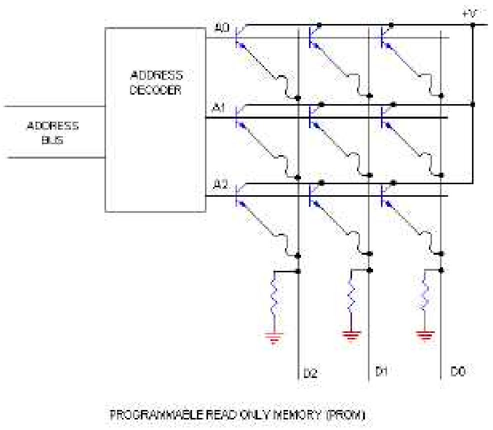


Figure 2-13

PROMs can only be programmed once. They are more fragile than ROMs. A jolt of static electricity can easily cause fuses in the PROM to burn out, changing essential bits from 1 to 0. But blank PROMs are inexpensive and are good for prototyping the data for a ROM before committing to the costly ROM fabrication process.

EPROM

Working with ROMs and PROMs can be a wasteful business. Even though they are inexpensive per chip, the cost can add up over time. Erasable programmable read-only memory (EPROM) addresses this issue. EPROM chips can be rewritten many times. Erasing an EPROM requires a special tool that emits a certain frequency of ultraviolet (UV) light. EPROMs are configured using an EPROM programmer that provides voltage at specified levels depending on the type of EPROM used.

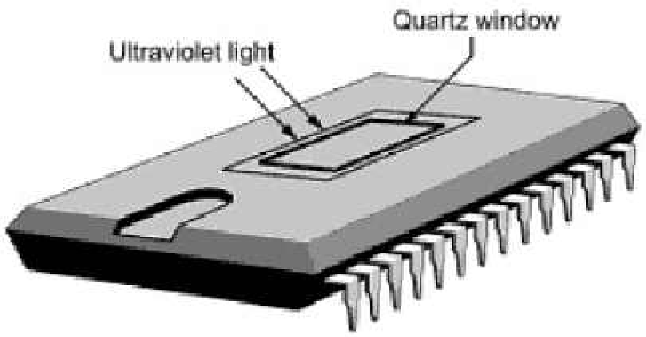


Figure 2-14

The EPROM has a grid of columns and rows and the cell at each intersection has two transistors. The two transistors are separated from each other by a thin oxide layer. One of the transistors is known as the floating gate and the other as the control gate. The floating gate's only link to the row (word line) is through the control gate. As long as this link is in place, the cell has a value of 1. To change the value to 0 requires a process called Fowler-Nordheim tunneling.

Tunneling is used to alter the placement of electrons in the floating gate. Tunneling creates an avalanche discharge of electrons, which have enough energy to pass through the insulating oxide layer and accumulate on the gate electrode. When the high voltage is removed, the electrons are trapped on the electrode. Because of the high insulation value of the silicon oxide surrounding the gate, the stored charge cannot readily leak away and the data can be retained for decades. An electrical charge, usually 10 to 13 volts, is applied to the floating gate. The charge comes from the column (bit line), enters the floating gate and drains to a ground.

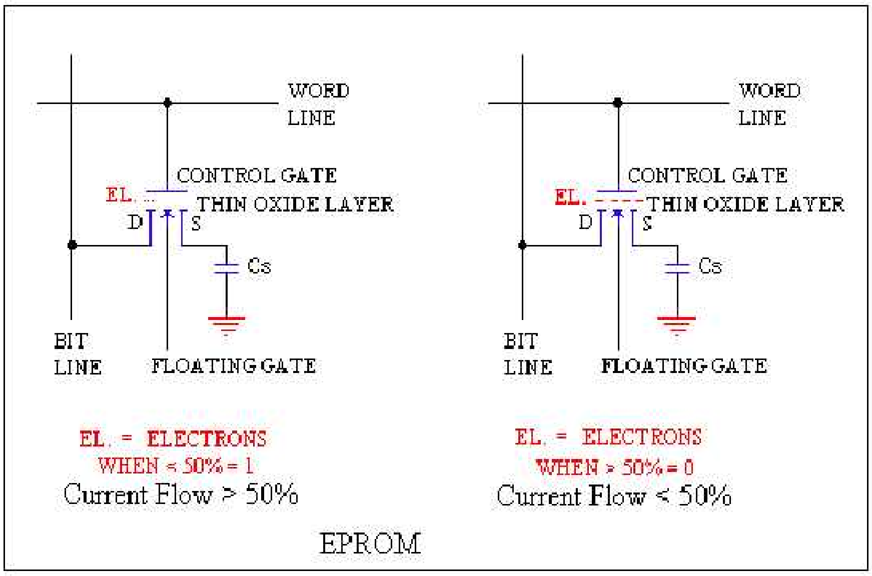


Figure 2-15

This charge causes the floating-gate transistor to act like an electron gun. The excited electrons are pushed through and trapped on the other side of the thin oxide layer, giving it a negative charge. These negatively charged electrons act as a barrier between the control gate and the floating gate. A device called a cell sensor monitors the level of the charge passing through the floating gate. If the flow through the gate is greater than 50 percent of the charge, it has a value of 1, when the charge passing through drops below the 50-percent threshold, the value changes to 0. A blank EPROM has all of the gates fully open, giving each cell a value of 1.

To rewrite an EPROM, you must erase it first. To erase it, you must supply a level of energy strong enough to break through the negative electrons blocking the floating gate. In a standard EPROM, this is best accomplished with UV light at a wavelength of 253.7 nanometers (2537 angstroms). Because this particular frequency will not penetrate most plastics or glasses, each EPROM chip has a quartz window on top of it. The EPROM must be very close to the eraser's light source, within an inch or two, to work properly.

An EPROM eraser is not selective; it will erase the entire EPROM. The EPROM must be removed from the device it is in and placed under the UV light of the EPROM eraser for several minutes. An EPROM that is left under too long can become over-erased. In such a case, the EPROM's floating gates are charged to the point that they are unable to hold the electrons at all.

EEPROMs and Flash Memory

Though EPROMs are a big step up from PROMs in terms of reusability, they still require dedicated equipment and a labor-intensive process to remove and reinstall them each time a change is necessary. Also, changes cannot be made incrementally to an EPROM; the whole chip must be erased. Electrically erasable programmable read-only memory (EEPROM) chips remove the biggest drawbacks of EPROMs. In EEPROMs:

1. The chip does not have to remove to be rewritten.
2. The entire chip does not have to be completely erased to change a specific portion of it.
3. Changing the contents does not require additional dedicated equipment.

Instead of using UV light, you can return the electrons in the cells of an EEPROM to normal with the localized application of an electric field to each cell. This erases the targeted cells of

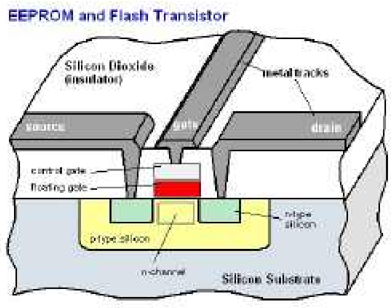


Figure 2-16

the EEPROM, which can then be rewritten. EEPROMs are changed 1 byte at a time, which makes them versatile but slow. In fact, EEPROM chips are too slow to use in many products that make quick changes to the data stored on the chip.

Manufacturers responded to this limitation with Flash memory, a type of EEPROM that uses in-circuit wiring to erase by applying an electrical field to the entire chip or to predetermined sections of the chip called blocks. This erases the targeted area of the chip, which can then be rewritten. Flash memory works much faster than traditional EEPROMs because instead of erasing one byte at a time, it erases a block or the entire chip, and then rewrites it. The electrons in the cells of a Flash-memory chip can be returned to normal ("1") by the application of an electric field, a higher-voltage charge.

Check your progress

Q1. Differentiate between ROM and PROM

Q2. Explain EPROM, EEPROM, Flash.

2.5 Serial Access Memory (SAM)

Serial access memory (SAM) stores data as a series of memory cells that can only be accessed sequentially (like a cassette tape). If the data is not in the current location, each memory cell is checked until the needed data is found. SAM works very well for memory buffers, where the data is normally stored in the order in which it will be used (a good example is the texture buffer memory on a video card)

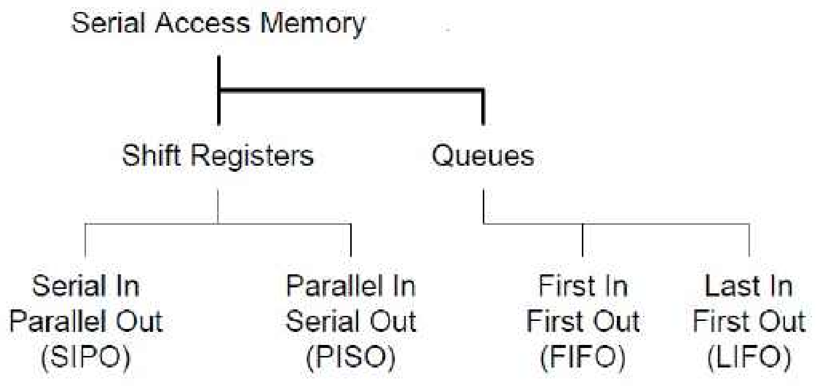

Serial Access Memory
Shift Registers
Queues
Serial In Parallel Out (SIPO)
Parallel In Serial Out (PISO)
First In First Out (FIFO)
Last In First Out (LIFO)

Figure 2-17

Shift registers, like counters, are a form of sequential logic. Sequential logic, unlike combinational logic is not only affected by the present inputs, but also, by the prior history. In other words, sequential logic remembers past events.

Shift registers produce a discrete delay of a digital signal or waveform. A waveform synchronized to a clock, a repeating square wave, is delayed by "n" discrete clock times, and where "n" is the number of shift register stages. Thus, a four stage shift register delays "data in" by four clocks to "data out". The stages in a shift register are delay stages, typically type "D" Flip-Flops or type "JK" Flip-flops.

Formerly, very long (several hundred stages) shift registers served as digital memory. This obsolete application is reminiscent of the acoustic mercury delay lines used as early computer memory.

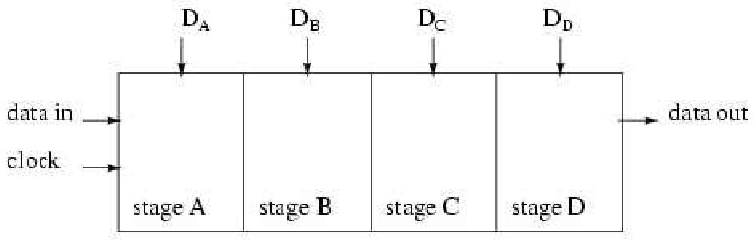
Serial data transmission, over a distance of meters to kilometers, uses shift registers to convert parallel data to serial form. Serial data communications replaces many slow parallel data wires with a single serial high speed circuit.

Serial data over shorter distances of tens of centimeters, uses shift registers to get data into and out of microprocessors. Numerous peripherals, including analog to digital converters, digital to analog converters, display drivers, and memory, use shift registers to reduce the amount of wiring in circuit boards.

Some specialized counter circuits actually use shift registers to generate repeating waveforms. Longer shift registers, with the help of feedback generate patterns so long that they look like random noise, pseudo-noise.

Basic shift registers are classified by structure according to the following types:

- Parallel-in/serial-out
- Serial-in/parallel-out

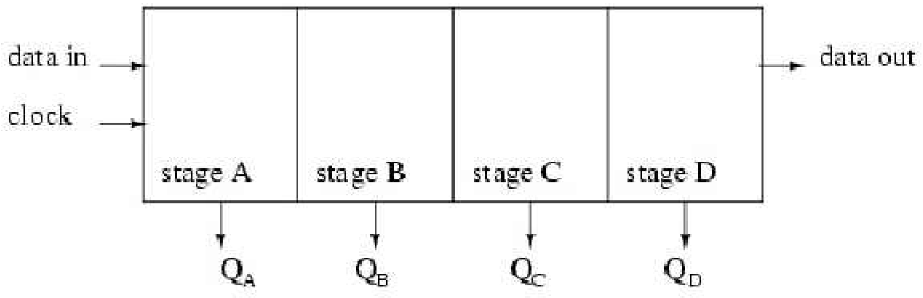


Parallel-in, serial-out shift register with 4-stages

Figure 2-18

Data is loaded into all stages at once of a parallel-in/serial-out shift register. The data is then shifted out via "data out" by clock pulses. Since a 4- stage shift register is shown above, four clock pulses are required to shift out all of the data. In the diagram above, stage D data will be present at the "data out" up until the first clock pulse; stage C data will be present at "data out" between the first clock and the second clock pulse; stage B data will be present between the second clock and the third clock; and stage A data will be present between the third and the fourth clock. After the fourth clock pulse and thereafter, successive bits of "data in" should appear at "data out" of the shift register after a delay of four clock pulses.

If four switches were connected to DA through DD, the status could be read into a microprocessor using only one data pin and a clock pin. Since adding more switches would require no additional pins, this approach looks attractive for many inputs.



Serial-in, parallel-out shift register with 4-stages

Figure 2-19

Above, four data bits will be shifted in from "data in" by four clock pulses and be available at QA through QD for driving external circuitry such as LEDs, lamps, relay drivers, and horns. After the first clock, the data at "data in" appears at QA. After the second clock, The old QA data appears at QB; QA receives next data from "data in". After the third clock, QB data is at QC. After the fourth clock, QC data is at QD. This stage contains the data first present at "data in". The shift register should now contain four data bits.

Queue

A **queue** is a particular kind of collection in which the entities in the collection are kept in order and the principal (or only) operations on the collection are the addition of entities to the rear terminal position and removal of entities from the front terminal position. This makes the queue a First-In-First-Out (FIFO) data structure. In a FIFO data structure, the first element added to the queue will be the first one to be removed. This is equivalent to the requirement that once an element is added, all elements that were added before have to be removed before the new element can be invoked. A queue is an example of a linear data structure.

Queues provide services in computer science, transport, and operations research where various entities such as data, objects, persons, or events are stored and held to be processed later. In these contexts, the queue performs the function of a buffer.

Queues are common in computer programs, where they are implemented as data structures coupled with access routines, as an abstract data structure or in object-oriented languages as classes. Common implementations are circular buffers and linked lists.

First in First Out

FIFO is an acronym for **First In, First Out**, which is an abstraction related to ways of organizing and manipulation of data relative to time and prioritization. This expression describes the principle of a queue processing technique or servicing conflicting demands by ordering process by first-come, first-served (**FCFS**) behavior: where the persons leave the queue in the order they arrive, or waiting one's turn at a traffic control signal.

FCFS is also the jargon term for the FIFO operating system scheduling algorithm, which gives every process CPU time in the order they come. In the broader sense, the abstraction LIFO or Last-In-First-Out is the opposite of the abstraction FIFO organization. The difference perhaps is clearest with considering the less commonly used synonym of LIFO, FILO (meaning First-In-Last-Out). In essence, both are specific cases of a more generalized list (which could be accessed anywhere). The difference is not in the list (data), but in the rules for accessing the content. One sub-type adds to one end, and takes off from the other, its opposite takes and puts things only on one end.

A slang variation on an ad-hoc approach to removing items from the queue has been coined as OFFO, which stands for On-Fire-First-Out. A priority queue is a variation on the queue which does not qualify for the name FIFO, because it is not accurately descriptive of that data structure's behavior. Queuing theory encompasses the more general concept of queue, as well as interactions between strict-FIFO queues.

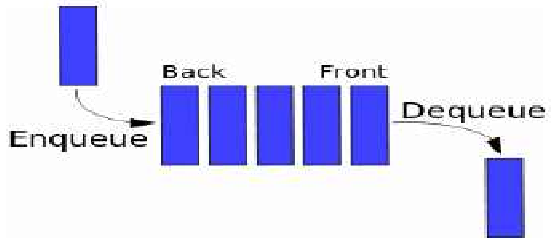


Figure 2-19

Last in First Out

LIFO is an acronym that stands for **last in, first out.** In computer science and queuing theory this refers to the way items stored in some types of data structures are processed. By definition, in a LIFO structured linear list, elements can be added or taken off from only one end, called the "*top*". A LIFO structure can be illustrated with the example of a stack of trays. The last tray to be placed on top is also the first to be taken off the top.

The term in computing generally refers to the abstract principles of list processing and temporary storage, particularly when there is a need to access the data in limited amounts, and in a certain order. LIFO is most used in cases where the last data added to the structure must be the first data to be removed or evaluated. A useful analogy is of the office worker: a person can only handle one page at a time, so the top piece of paper added to a pile is the first off; parallel to limitations such as data bus width and the fact that one can only manipulate a single binary data address in a computer at a time. The abstract LIFO mechanism, when applied to computing inevitably devolves to the real data structures implemented as stacks whose eponymous relation to the "stack of paper", "stack of plates" should be obvious. Other names for the device are "Push down list" and "piles". The term **FILO** ("first in, last out") can be used synonymously, as the term emphasizes that early additions to the list need to wait until they rise to the LIFO structure "top" to be accessed. The term **LCFS** ("last come, first served") is sometimes used in queuing theory. The difference between a generalized list, an array, queue, or stack, is defined by the rules enforced and used to access the mechanism. In any event, an LIFO structure is accessed in opposite order to a queue: "There are certain frequent situations in computer science when one wants to restrict insertions and deletions so that they can only take place at the beginning or end of the list, not in the middle. Two of the data structures useful in such situations are *stacks* and *queues*."

Check your progress

Q1. What do you mean by Serial access memory? How many types of serial access memory available?

2.6 Summary

In this unit you learnt about Memory –Introduction, Memory Cell, Memory Organization, Read Only Memory, and Serial Access Memory.

- Computer memory is the storage space in computer where data is to be processed and instructions required for processing are stored.

- Primary computer memory is made up of memory cells, where each memory cell contains exactly one number.
- A typical computer has millions of memory cells, and every memory cell has a name.
- A memory consists of a large number of memory cells, there has to be a way to keep track of which cell is which.
- Registers are memories located within the Central Processing Unit (CPU).

2.7 Review Questions

Q1. What is the role of memory in computer? Explain in detail.

Q2. Define the term 'Memory Cell' in detail with example.

Q3. What is the role of memory organization in the computer system? Explain with example.

Q4. Define Read Only Memory in detail.

Q5. Define Serial Access Memory with example.

UNIT-III

Memory in Physical Devices

Structure

3.0 Introduction

3.1 Physical Devices Used to Construct Memories

3.2 Magnetic Hard Disk

3.3 Summary

3.4 Review Questions

3.0 Introduction

In this unit we focused on memory in physical devices. There are four sections in this unit. In the first section i.e. Sec. 3.1 you will learn about Physical Devices Used to Construct Memories. As you know *computing* any device, such as a disk, printer, modem, or screen, concerned with input/output, storage, etc. Often shortened to: peripheral devices. As you also know that the memory, a memory is just like a human brain. It is used to store data and instruction. Computer memory is the storage space in computer where data is to be processed and instructions required for processing are stored. The memory is divided into large number of small parts. Each part is called a cell. Each location or cell has a unique address which varies from zero to memory size minus one. For example if computer has 64k words, then this memory unit has 64 * 1024 = 65536 memory location. The address of these locations varies from 0 to 65535. In Sec. 3.2 you will know about magnetic hard disk in detail. In Sec. 3.3 and 3.4 you will find summary and review questions respectively.

Objectives

After studying this unit you should be able to:

- Define how to construct memories.
- Express magnetic hard disk

3.1 Physical Devices Used to Construct Memories

There are several different types of memory in computer systems. There are physical memory and virtual memory, and neither should be confused with hard drive space. Physical memory is the random access memory, or RAM, in the system. The amount of RAM in a system is modified by a paging file set aside on your hard drive known as the virtual memory, which is used in the same way as RAM.

What Is RAM

- RAM is a type of data storage device that the central processing unit, or CPU, uses to access information in any order. RAM is volatile memory and is lost when the system is powered down. The more RAM a computer has the less it will have to access the hard drive for data. The RAM slots in the motherboard are directly connected to the CPU through the system bus to allow for the fastest communication between the CPU and the RAM.

Figure 3-1

Where Is RAM

- There are usually two to four slots in computer motherboards for RAM chips to be installed. RAM varies in their megahertz rating, or speed, and the number of pins on the card that fit into the slots on the motherboard. The RAM must be both the same number of pins and a speed that is compatible with the particular motherboard. RAM also comes in different sizes in regard to the amount of memory the RAM card has.

Adding RAM

- The best way to upgrade your RAM is to open the case on your system and write down the brand and model of the motherboard, printed somewhere on the main board itself. Remove the RAM cards from your system and bring them with you to the computer store. With the RAM cards and brand and model of the motherboard, the computer salesman can look up the type of RAM you can use and provide you with choices as to what type and how much you can add.

There are two basic types of RAM:

(i) Dynamic Ram

(ii) Static RAM

Dynamic RAM: loses its stored information in a very short time (for millisecond) even when power supply is on. D-RAM's are cheaper & lower.

Similar to a microprocessor chip it is an Integrated Circuit (IC) made of millions of transistors and capacitors.

In the most common form of computer memory, Dynamic Memory Cell represents a single bit of data. The capacitor holds the bit of information – a 0 or a 1. The transistor acts as a switch that lets the control circuitry on the memory chip read the capacitor or change its state. A capacitor is like a small bucket that is able to store electrons. To store a 1 in the memory cell, the bucket is filled with electrons.

To store a 0, it is emptied. The problem with the capacitor's bucket is that it has a leak. In a matter of a few milliseconds a full bucket becomes empty. Therefore, for dynamic memory to work, either the CPU or the Memory Controller has to come along and recharge all of the capacitors holding it before they discharge. To do this, the memory controller reads the memory and then writes it right back. This refresh operation happens automatically thousands of times per second.

This refresh operation is where dynamic RAM gets its name. Dynamic RAM has to be dynamically refreshed all of the time or it forgets what it is holding. The downside of all of this refreshing is that it takes time and slows down the memory.

Static RAM: - uses a completely different technology. S-RAM retains stored information only as long as the power supply is on. Static RAM's are costlier and consume more power. They have higher speed than D-RAMs. They store information in Hip-Hope.

In static RAM, a form of flip-flop holds each bit of memory. A flip-flop for a memory cell takes four or six transistors along with some wiring, but never has to be refreshed. This makes static RAM significantly faster than dynamic RAM. However, because it has more parts, a static memory cell takes up a lot more space on a chip than a dynamic memory cell. Therefore, you get less memory per chip, and that makes static RAM a lot more expensive. Static RAM is fast and expensive, and dynamic RAM is less expensive and slower. Static RAM is used to create the CPU's speed sensitive cache, while dynamic RAM forms the larger system RAM space.

Some other RAMS are:

(a) EDO (Extended Data Output) RAM: In an EDO RAMs, any memory location can be accessed Stores 256 bytes of data information into latches. The latches hold next 256 bytes of information so that in most programs, which are sequentially executed, the data are available without wait states.

(b) SDRAM (Synchronous DRAMS), SGRAMs (Synchronous Graphic RAMs) These RAM chips use the same clock rate as CPU uses. They transfer data when the CPU expects them to be ready.

(c) DDR-SDRAM (Double Data Rate – SDRAM): This RAM transfers data on both edges of the clock. Therefore the transfer rate of the data becomes doubles.

Virtual Memory

Virtual memory is a common part of most operating systems on desktop computers. It has become so common because it provides a big benefit for users at a very low cost.

In this article, you will learn exactly what virtual memory is, what your computer uses it for and how to configure it on your own machine to achieve optimal performance.

Most computers today have something like 32 or 64 megabytes of RAM available for the CPU to use (see How RAM Works for details on RAM). Unfortunately, that amount of RAM is not enough to run all of the programs that most users expect to run at once.

For example, if you load the operating system, an e-mail program, a Web browser and word processor into RAM simultaneously, 32megabytes is not enough to hold it all. If there were no such thing as virtual memory, then once you filled up the available RAM your computer would have to say, "Sorry, you cannot load any more applications. Please close another application to load a new one." With virtual memory, what the computer can do is look at RAM for areas that have not been used recently and copy them onto the hard disk. This frees up space in RAM to load the new application.

Because this **copying** happens automatically, you don't even know it is happening, and it makes your computer feel like is has unlimited RAM space even though it only has 32 megabytes installed. Because hard disk space is so much cheaper than RAM chips, it also has a nice economic benefit.

The read/write speed of a hard drive is much slower than RAM, and the technology of a hard drive is not geared toward accessing small pieces of data at a time. If your system has to rely too heavily on virtual memory, you will notice a significant performance drop. The key is to have enough RAM to handle everything you tend to work on simultaneously -- then, the only time you "feel" the slowness of virtual memory is when there's a slight pause when you're changing tasks. When that's the case, virtual memory is perfect.

When it is not the case, the operating system has to constantly swap information back and forth between RAM and the hard disk. This is called **thrashing**, and it can make your computer feel incredibly slow.

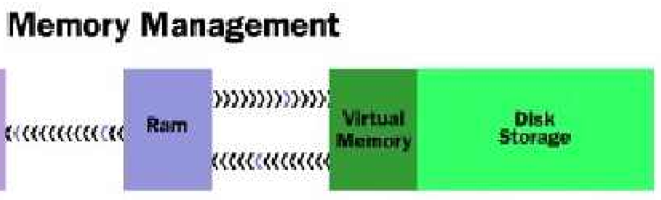


Figure 3-2

The area of the hard disk that stores the RAM image is called a **page file**. It holds **pages** of RAM on the hard disk, and the operating system moves data back and forth between the pages file and RAM. On a Windows machine, page files have a .SWP extension.

Check your progress

Q1. What is a RAM? Define the working of RAM.

Q2. Compare Static and Dynamic RAM.

Q3. What is the use of virtual memory?

3.2 Magnetic Hard Disk

A **hard disk drive (HDD)** is a data storage device used for storing and retrieving digital information using rapidly rotating discs (platters) coated with magnetic material. An HDD retains its data even when powered off. Data is read in a random-access manner, meaning individual blocks of data can be stored or retrieved in any order rather than sequentially. An HDD consists of one or more rigid ("hard") rapidly rotating discs (platters) with magnetic heads arranged on a moving actuator arm to read and write data to the surfaces.

Introduced by IBM in 1956, HDDs became the dominant secondary storage device for general purpose computers by the early 1960s. Continuously improved, HDDs have maintained this position into the modern era of servers and personal computers. More than 200 companies have produced HDD units, though most current units are manufactured by Seagate, Toshiba and Western Digital. Worldwide revenues for HDDs shipments are expected to reach $33 billion in 2013, a decrease of about 12% from $37.8 billion in 2012.

The primary characteristics of an HDD are its capacity and performance. Capacity is specified in unit prefixes corresponding to powers of 1000: a 1-terabyte (TB) drive has a capacity of 1,000 gigabytes (GB; where 1 gigabyte = 1 billion bytes). Typically, some of an HDD's capacity is unavailable to the user because it is used by the file system and the computer operating system, and possibly inbuilt redundancy for error correction and recovery. Performance is specified by the time to move the heads to a file (Average Access Time) plus the time it takes for the file to move under its head (average latency, a function of the physical rotational speed in revolutions per minute) and the speed at which the file is transmitted (data rate).

The two most common form factors for modern HDDs are 3.5-inch in desktop computers and 2.5-inch in laptops. HDDs are connected to systems by standard interface cables such as SATA (Serial ATA), USB or SAS (Serial attached SCSI) cables.

As of 2012, the primary competing technology for secondary storage is flash memory in the form of solid-state drives (SSDs). HDDs are expected to remain the dominant medium for secondary storage due to predicted continuing advantages in recording capacity and price per unit of storage; but SSDs are replacing HDDs where speed, power consumption and durability are more important considerations than price and capacity.

Magnetic recording

An HDD records data by magnetizing a thin film of ferromagnetic material on a disk. Sequential changes in the direction of magnetization represent binary data bits. The data is read from the disk by detecting the transitions in magnetization. User data is encoded using an encoding scheme, such as run-length limited encoding, which determines how the data is represented by the magnetic transitions.

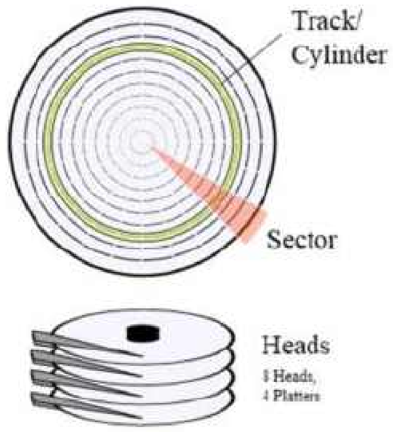


Figure 3-3

A typical HDD design consists of a *spindle* that holds flat circular disks, also called platters, which hold the recorded data. The platters are made from a non-magnetic material, usually aluminum alloy, glass, or ceramic, and are coated with a shallow layer of magnetic material typically 10–20 nm in depth, with an outer layer of carbon for protection. For reference, a standard piece of copy paper is 0.07–0.18 millimeter (70,000–180,000 nm).

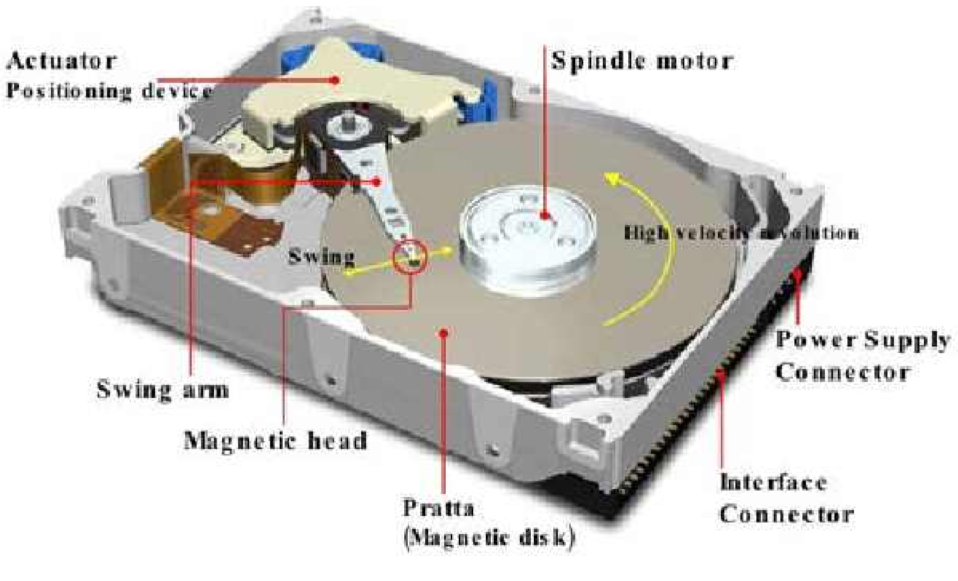


Figure 3-4

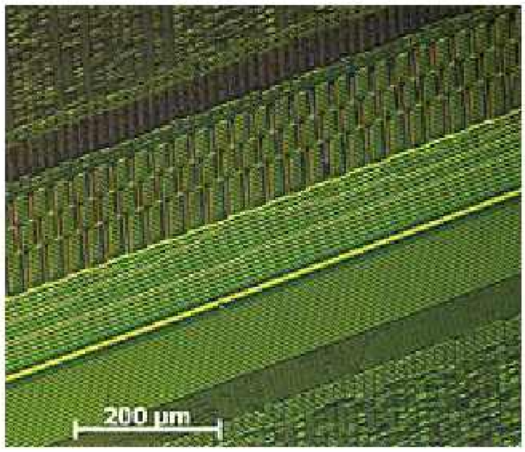


Figure 3-5

Recording of single magnetizations of bits on a 200MB HDD-platter (recording made visible using CMOS-Mag View).

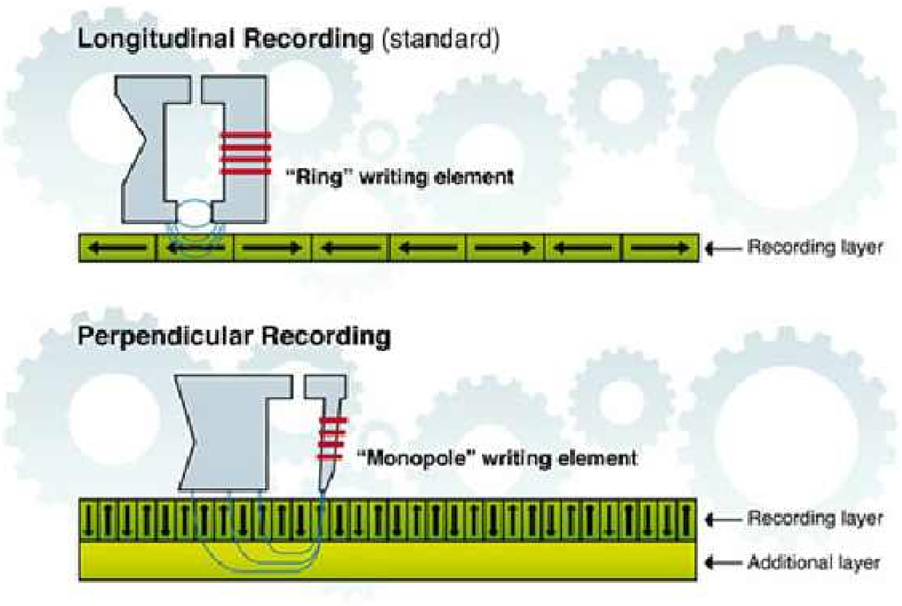


Figure 3-6

Longitudinal recording (standard) & perpendicular recording diagram

The platters in contemporary HDDs are spun at speeds varying from 4,200 rpm in energy-efficient portable devices, to 15,000 rpm for high performance servers. The first HDDs spun at 1,200 rpm and, for many years, 3,600 rpm was the norm. Today, the platters in most consumer HDDs spin in the range of 5,400 rpm to 7,200 rpm.

Information is written to and read from a platter as it rotates past devices called read-and-write heads that operate very close (often tens of nanometers) over the magnetic surface. The read-and-write head is used to detect and modify the magnetization of the material immediately under it.

In modern drives there is one head for each magnetic platter surface on the spindle, mounted on a common arm. An actuator arm (or access arm) moves the heads on an arc (roughly radially) across the platters as they spin, allowing each head to access almost the entire surface of the platter as it spins. The arm is moved using a voice coil actuator or in some older designs a stepper motor. Early hard disk drives wrote data at some constant bits per second, resulting in all tracks having the same amount of data per track but modern drives (since the 1990s) use zone bit recording -- increasing the write speed from inner to outer zone and thereby storing more data per track in the outer zones.

In modern drives, the small size of the magnetic regions creates the danger that their magnetic state might be lost because of thermal effects. To counter this, the platters are coated with two parallel magnetic layers, separated by a 3-atom layer of the non-magnetic element ruthenium, and the two layers are magnetized in opposite orientation, thus reinforcing each other. Another technology used to overcome thermal effects to allow greater recording densities is perpendicular recording, first shipped in 2005, and as of 2007 the technology was used in many HDDs.

Components

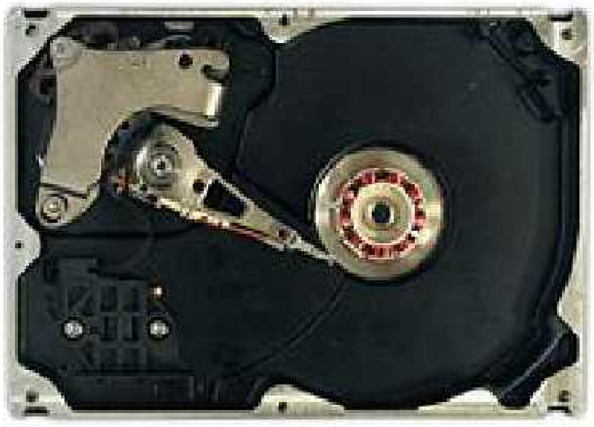


Figure 3-7

HDD with disks and motor hub removed exposing copper colored stator coils surrounding a bearing in the center of the spindle motor, Orange stripe along the side of the arm is thin printed-circuit cable, spindle bearing is in the center and the actuator is in the upper left

A typical HDD has two electric motors; a spindle motor that spins the disks and an actuator (motor) that positions the read/write head assembly across the spinning disks. The disk motor has an external rotor attached to the disks; the stator windings are fixed in place. Opposite the actuator at the end of the head support arm is the read-write head; thin printed-circuit cables connect the read-write heads to amplifier electronics mounted at the pivot of the actuator. The head support arm is very light, but also stiff; in modern drives, acceleration at the head reaches 550 g.

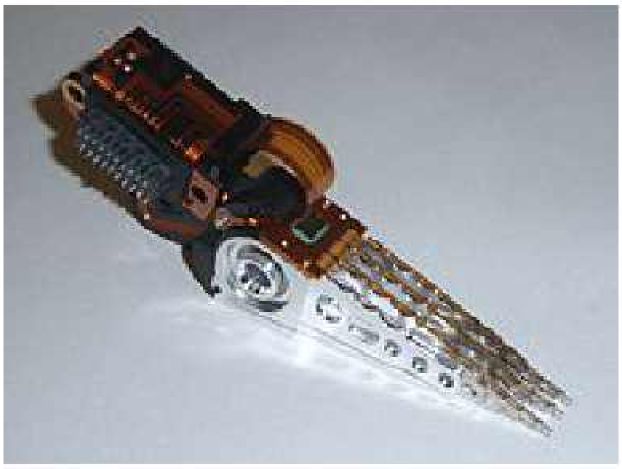


Figure 3-8

Head stack with an actuator coil on the left and read/write heads on the right

The actuator is a permanent magnet and moving coil motor that swings the heads to the desired position. A metal plate supports a squat neodymium-iron-boron (NIB) high-flux magnet. Beneath this plate is the moving coil, often referred to as the voice coil by analogy to the coil in loudspeakers, which is attached to the actuator hub, and beneath that is a second NIB magnet, mounted on the bottom plate of the motor (some drives only have one magnet).

The voice coil itself is shaped rather like an arrowhead, and made of doubly coated copper magnet wire. The inner layer is insulation, and the outer is thermoplastic, which bonds the coil together after it is wound on a form, making it self-supporting. The portions of the coil along the two sides of the arrowhead (which point to the actuator bearing center) interact with the magnetic field, developing a tangential force that rotates the actuator. Current flowing radially outward along one side of the arrowhead and radially inward on the other produces the tangential force. If the magnetic field were uniform, each side would generate opposing forces that would cancel each other out. Therefore the surface of the magnet is half N pole, half S pole, with the radial dividing line in the middle, causing the two sides of the coil to see opposite magnetic fields and produce forces that add instead of canceling. Currents along the top and bottom of the coil produce radial forces that do not rotate the head.

The HDD's electronics control the movement of the actuator and the rotation of the disk, and perform reads and writes on demand from the disk controller. Feedback of the drive electronics is accomplished by means of special segments of the disk dedicated to servo feedback. These are either complete concentric circles (in the case of dedicated servo technology), or segments interspersed with real data (in the case of embedded servo technology). The servo feedback optimizes the signal to noise ratio of the GMR sensors by adjusting the voice-coil of the actuated arm. The spinning of the disk also uses a servo motor. Modern disk firmware is capable of scheduling reads and writes efficiently on the platter surfaces and remapping sectors of the media which have failed.

Error handling

Modern drives make extensive use of error correction codes (ECCs), particularly Reed–Solomon error correction. These techniques store extra bits, determined by mathematical formulas, for each block of data; the extra bits allow many errors to be corrected invisibly. The extra bits themselves take up space on the HDD, but allow higher recording densities to be employed without causing uncorrectable errors, resulting in much larger storage capacity. In the newest drives of 2009, low-density parity-check codes (LDPC) were supplanting Reed-Solomon; LDPC codes enable performance close to the Shannon Limit and thus provide the highest storage density available.

Typical HDDs attempt to "remap" the data in a physical sector that is failing to a spare physical sector—hopefully while the errors in the bad sector are still few enough that the ECC can recover the data without loss. The S.M.A.R.T-Self-Monitoring, Analysis and Reporting Technology system counts the total number of errors in the entire HDD fixed by ECC and the total number of re-mappings, as the occurrence of many such errors may predict HDD failure.

Capacity

The capacity of an HDD reported to an end user by the operating system is less than the amount stated by a drive or system manufacturer due to amongst other things, different units of measuring capacity, capacity consumed by the file system and/or redundancy.

Calculation

Because modern disk drives appear to their interface as a contiguous set of logical blocks their gross capacity can be calculated by multiplying the number of blocks by the size of the block. This information is available from the manufacturer's specification and from the drive itself through use of special utilities invoking low level commands.

The gross capacity of older HDDs can be calculated by multiplying for each zone of the drive the number of cylinders by the number of heads by the number of sectors/zone by the number of bytes/sector (most commonly 512) and then summing the totals for all zones. Some modern SATA drives will also report cylinder-head-sector (C/H/S) values to the CPU but they are no longer actual physical parameters since the reported numbers are constrained by historic operating-system interfaces.

The old C/H/S scheme has been replaced by logical block addressing. In some cases, to try to "force-fit" the C/H/S scheme to large-capacity drives, the number of heads was given as 64, although no modern drive has anywhere near 32 platters.

Redundancy

In modern HDDs, spare capacity for defect management is not included in the published capacity; however in many early HDDs a certain number of sectors were reserved for spares, thereby reducing capacity available to end users.

In some systems, there may be hidden partitions used for system recovery that reduce the capacity available to the end user.

For RAID subsystems, data integrity and fault-tolerance requirements also reduce the realized capacity. For example, a RAID1 subsystem will be about half the total capacity as a result of data mirroring. RAID5 subsystems with x drives would lose 1/x of capacity to parity. RAID subsystems are multiple drives that appear to be one drive or more drives to the user, but provides a great deal of fault-tolerance. Most RAID vendors use some form of checksums to improve data integrity at the block level. For many vendors, this involves using HDDs with sectors of 520 bytes per sector to contain 512 bytes of user data and eight checksum bytes or using separate 512-byte sectors for the checksum data.

File system use

The presentation of an HDD to its host is determined by its controller. This may differ substantially from the drive's native interface particularly in mainframes or servers. Modern HDDs, such as SAS and SATA drives, appear at their interfaces as a contiguous set of logical blocks; typically 512 bytes long but the industry is in the process of changing to 4,096-byte logical blocks; see Advanced Format.

The process of initializing these logical blocks on the physical disk platters is called *low level formatting* which is usually performed at the factory and is not normally changed in the field.

High level formatting then writes the file system structures into selected logical blocks to make the remaining logical blocks available to the host OS and its applications. The operating system file system uses some of the disk space to organize files on the disk, recording their file names and the sequence of disk areas that represent the file. Examples of data structures stored on disk to retrieve files include the file allocation table (FAT) in the MS-DOS file system and inodes in many UNIX file systems, as well as other operating system data structures. As a consequence not all the space on an HDD is available for user files. This file system overhead is usually less than 1% on drives larger than 100 MB.

Form factors

Past and present HDD form factors

Form factor	**Status**	**Length [mm]**	**Width [mm]**	**Height [mm]**	**Largest capacity**	**Platters (max)**	**Capacity Per platter [GB]**
3.5"	Current	146	101.6	19 or 25.4	4 TB (2011)	5	1000
2.5"	Current	100	69.85	5, 7, 9.5,12.5, or 15	2 TB (2012)	4	69
1.8"	Current	71	54	5 or 8	320 GB (2009)	2	220
8"	Obsolete	362	241.3	117.5			
5.25"	Obsolete	203	146	82.6	47 GB[1] (1998)	14	3.36

5.25"	Obsolete	203	146	41.4	19.3 GB[(1998)	4	4.83
1.3"	Obsolete		43		40 GB (2007)	1	40
1" (CFII/ZIF/IDE-Flex)	Obsolete		42		20 GB (2006)	1	20
0.85"	Obsolete	32	24	5	8 GB (2004)	1	8

Mainframe and minicomputer hard disks were of widely varying dimensions, typically in free standing cabinets the size of washing machines or designed to fit a 19" rack. In 1962, IBM introduced its model 1311 disk, which used 14 inch (nominal size) platters. This became a standard size for mainframe and minicomputer drives for many years. Such large platters were never used with microprocessor-based systems.

With increasing sales of microcomputers having built in floppy-disk drives (FDDs), HDDs that would fit to the FDD mountings became desirable. Thus HDD *Form factors* initially followed those of 8-inch, 5.25-inch, and 3.5-inch floppy disk drives. Because there were no smaller floppy disk drives, smaller HDD form factors developed from product offerings or industry standards.

8 inch

9.5 in × 4.624 in × 14.25 in (241.3 mm × 117.5 mm ×362 mm). In 1979, Shugart Associates' SA1000 was the first form factor compatible HDD, having the same dimensions and a compatible interface to the 8" FDD.

5.25 inch

5.75 in × 3.25 in × 8 in (146.1 mm × 82.55 mm × 203 mm). This smaller form factor, first used in an HDD by Seagate in 1980, was the same size as full-height 5¼-inch-diameter (130 mm) FDD, 3.25-inches high. This is twice as high as "half height"; i.e., 1.63 in (41.4 mm). Most desktop models of drives for optical 120 mm disks (DVD, CD) use the half height 5¼" dimension, but it fell out of fashion for HDDs. The Quantum Bigfoot HDD was the last to use it in the late 1990s, with "low-profile" (≈25 mm) and "ultra-low-profile" (≈20 mm) high versions.

3.5 inch

4 in × 1 in × 5.75 in (101.6 mm × 25.4 mm × 146 mm) = 376.77344 cm^3. This smaller form factor is similar to that used in an HDD by Rodime in 1983, which was the same size as the "half height" 3½" FDD, i.e., 1.63 inches high. Today, the 1-inch high ("slimline" or "low-profile") version of this form factor is the most popular form used in most desktops.

2.5 inch

2.75 in × 0.275–0.59 in × 3.945 in (69.85 mm × 7–15 mm × 100 mm) = 48.895–104.775 cm^3. This smaller form factor was introduced by PrairieTek in 1988; there is no corresponding FDD. It came to be widely used for HDDs in mobile devices (laptops, music players, etc.) and for solid-state drives (SSDs), by 2008 replacing some 3.5 inch enterprise-class drives. It is also used in the PlayStation 3 and Xbox 360[video game consoles. Drives 9.5 mm high became an unofficial standard for all except the largest-capacity laptop drives (usually having two platters inside); 12.5 mm-high drives, typically with three platters, are used for maximum capacity, but will not fit most laptop computers. Enterprise-class drives can have a height up to 15 mm. Seagate released a 7mm drive aimed at entry level laptops and high end netbooks in December 2009.

1.8 inch

54 mm × 8 mm × 71 mm = 30.672 cm^3. This form factor, originally introduced by Integral Peripherals in 1993, evolved into the ATA-7 LIF with dimensions as stated. For a time it was increasingly used in digital audio players and subnotebooks, but its popularity decreased to the point where this form factor is increasingly rare and only a small percentage of the overall market.

1 inch

42.8 mm × 5 mm × 36.4 mm, this form factor was introduced in 1999 as IBM's Microdrive to fit inside a CF Type II slot. Samsung calls the same form factor *"1.3 inch" drive* in its product literature.

0.85 inch

24 mm × 5 mm × 32 mm. Toshiba announced this form factor in January 2004for use in mobile phones and similar applications, including SD/MMC slot compatible HDDs optimized for video storage on 4G handsets. Toshiba manufactured a 4 GB (MK4001MTD) and an 8 GB (MK8003MTD) version and holds the Guinness World Record for the smallest HDD.

As of 2012, 2.5-inch and 3.5-inch hard disks were the most popular sizes.

By 2009 all manufacturers had discontinued the development of new products for the 1.3-inch, 1-inch and 0.85-inch form factors due to falling prices of flash memory, which has no moving parts.

While these sizes are customarily described by an approximately correct figure in inches, actual sizes have long been specified in millimeters.

Performance characteristics

Time to access data

The factors that limit the time to access the data on an HDD are mostly related to the mechanical nature of the rotating disks and moving heads. Seek time is a measure of how long it takes the head assembly to travel to the track of the disk that contains data. Rotational latency is incurred because the desired disk sector may not be directly under the head when data transfer is requested. These two delays are on the order of milliseconds each. The bit rate or data transfer rate (once the head is in the right position) creates delay which is a function of the number of blocks transferred; typically relatively small, but can be quite long with the transfer of large contiguous files. Delay may also occur if the drive disks are stopped to save energy.

An HDD's *Average Access Time* is its average Seek time which technically is the time to do all possible seeks divided by the number of all possible seeks, but in practice is determined by statistical methods or simply approximated as the time of a seek over one-third of the number of tracks.

Defragmentation is a procedure used to minimize delay in retrieving data by moving related items to physically proximate areas on the disk. Some computer operating systems perform defragmentation automatically. Although automatic defragmentation is intended to reduce access delays, performance will be temporarily reduced while the procedure is in progress.

Time to access data can be improved by increasing rotational speed (thus reducing latency) and/or by reducing the time spent seeking. Increasing areal density increases throughputs by increasing data rate and by increasing the amount of data under a set of heads, thereby potentially reducing seek activity for a given amount of data. Based on historic trends, analysts predict a future growth in HDD areal density (and therefore capacity) of about 40% per year. The time to access data has not kept up with throughput increases, which themselves have not kept up with growth in storage capacity.

Seek time

Average seek time ranges from 3 ms for high-end server drives, to 15 ms for mobile drives, with the most common mobile drives at about 12 ms and the most common desktop type typically being around 9 ms. The first HDD had an average seek time of about 600 ms; by the middle 1970s HDDs were available with seek times of about 25 ms Some early PC drives used a stepper motor to move the heads, and as a result had seek times as slow as 80–120 ms, but this was quickly improved by voice coil type actuation in the 1980s, reducing seek times to around 20 ms. Seek time has continued to improve slowly over time.

Some desktop and laptop computer systems allow the user to make a tradeoff between seek performance and drive noise. Faster seek rates typically require more energy usage to quickly move the heads across the platter, causing louder noises from the pivot bearing and greater device vibrations as the heads are rapidly accelerated during the start of the seek motion and decelerated at the end of the seek motion. Quiet operation reduces movement speed and acceleration rates, but at a cost of reduced seek performance.

Latency

Latency is the delay for the rotation of the disk to bring the required disk sector under the read-write mechanism. It depends on rotational speed of a disk, measured in revolutions per minute (rpm). Average rotational latency is shown in the table below, based on the statistical relation that the average latency in milliseconds for such a drive is one-half the rotational period.

Rotational speed [rpm]	**Average latency [ms]**
15,000	2
10,000	3

7,200	4.16
5,400	5.55
4,800	6.25

Data transfer rate

As of 2010, a typical 7,200-rpm desktop HDD has a sustained "disk-to-buffer" data transfer rate up to 1,030 Mbits/sec. This rate depends on the track location; the rate is higher for data on the outer tracks (where there are more data sectors per rotation) and lower toward the inner tracks (where there are fewer data sectors per rotation); and is generally somewhat higher for 10,000-rpm drives. A current widely used standard for the "buffer-to-computer" interface is 3.0 Gbit/s SATA, which can send about 300 megabyte/s (10-bit encoding) from the buffer to the computer, and thus is still comfortably ahead of today's disk-to-buffer transfer rates. Data transfer rate (read/write) can be measured by writing a large file to disk using special file generator tools, then reading back the file. Transfer rate can be influenced by file system fragmentation and the layout of the files.

HDD data transfer rate depends upon the rotational speed of the platters and the data recording density. Because heat and vibration limit rotational speed, advancing density becomes the main method to improve sequential transfer rates. Higher speeds require more power absorbed by the electric engine, which hence warms up more. While areal density advances by increasing both the number of tracks across the disk and the number of sectors per track, only the latter increases the data transfer rate for a given rpm. Since data transfer rate performance only tracks one of the two components of areal density, its performance improves at a lower rate.

Check your progress

Q1. Define the working of the magnetic hard disk.

Q2. Explain form factors of the magnetic hard disk.

3.3 Summary

In this unit you learnt Physical Devices Used to Construct Memories, Magnetic Hard Disk

- The amount of RAM in a system is modified by a paging file set aside on your hard drive known as the virtual memory.
- RAM is a type of data storage device that the central processing unit, or CPU, uses to access information in any order.

- RAM varies in their megahertz rating, or speed, and the number of pins on the card that fit into the slots on the motherboard.
- A capacitor is like a small bucket that is able to store electrons. To store a 1 in the memory cell, the bucket is filled with electrons.
- A flip-flop for a memory cell takes four or six transistors along with some wiring, but never has to be refreshed.

3.4 Review Questions

Q1. Why we use RAM in computer system? Elaborate your answer.

Q2. Define the term paging with example.

Q3. What do you mean by flip flop? Describe its working with example.

Q4. Write a short note on Virtual Memory.

Q5. Write a note on Hard disk drive with example.

UNIT-IV

Disk Drives

Structure

4.0 Introduction

4.1 Floppy Disk Drives

4.2 Compact Disk Read Only Memory

4.3 Magnetic Tape Drives

4.4 Summary

4.5 Review Questions

4.0 Introduction

In this unit you will learn all about the disk drives. In this unit there are five sections. In the first sections i.e. Sec. 4.1 you will know about floppy disk drives. As you have studied earlier about floppy disk drives and as you know a floppy disk drive (FDD), or floppy drive, is a hardware device that reads data storage information. It was invented in 1967 by a team at IBM and was one of the first types of hardware storage that could read/write a portable device. FDDs are used for reading and writing on removable floppy discs. Floppy disks are now outdated, and have been replaced by other storage devices such as USB and network file transfer. In Sec. 4.2 CD- Disk has introduced. A compact disc [sometimes spelled disk] (CD) is a small, portable, roundmedium made of molded polymer (close in size to the floppy disk) for electronically recording, storing, and playing back audio, video, text, and other information in digital form. Tape cartridges and CDs generally replaced the phonograph record for playing back music. At home, CDs have tended to replace the tape cartridge although the latter is still widely used in cars and portable playback devices. In Sec. 4.3 you will know about magnetic disk. In Sec. 4.4 and 4.5 you will find summary and review questions respectively.

Objectives

After studying this unit you should be able to:

- Define floppy disk drives
- Describe compact disk read only memory
- Express magnetic tape drives

4.1 Floppy Disk Drives

A **Floppy Disk Drive**, or **FDD** or **FD** for short, is a computer disk drive that enables a user to save data to removable diskettes. Although 8" disk drives were first made available in 1971, the first real disk drives used were the 5 1/4" floppy disk drives, which were later replaced with 3 1/2" floppy disk drives. Today, because of the limited capacity and reliability of floppy diskettes many computers no longer come equipped with floppy disk drives and are being replaced with CD-R, other writable discs, and flash drives.

Parts of the Floppy Disk

The floppy disk stores information from computers and has two main parts: the protective components and recording components.

Protective Components

The protective components consist of the housing and shutter/spring.

Housing

A square protective outer plastic shell with two halves protects the inner contents of the floppy.

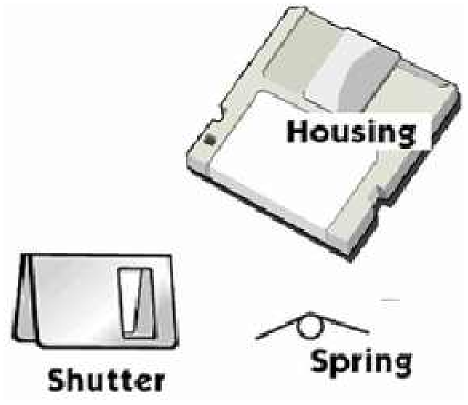


Figure 4-1

Shutter and spring

The shutter and the spring protect the information recorded on the disk. The shutter is a piece of metal over the housing. It slides over when inserted into the floppy drive, allowing access to the floppy and its contents. The spring closes the shutter once the disk is removed to keep fingerprints and dust off the floppy

Recording Components

Several components inside the floppy relate to the recording process.

Magnetic Disk

The magnetic disk is a round piece of plastic coated with iron oxide, which can be magnetized. When you save information to a disk, a recording head creates a magnetic pattern on the iron oxide. This pattern stores your words or pictures in a form that the computer can read the next time you put the disk in. However, if the write-protect tab is open, you cannot save data.

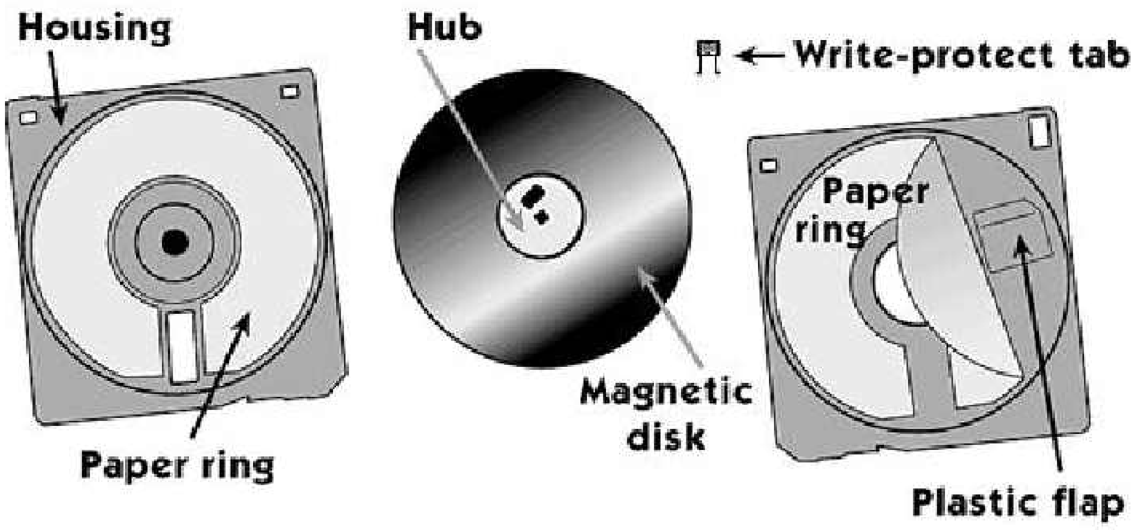


Figure 4-2

Write-Protect Tab

This little plastic rectangle is in the upper right corner of most disks. It slides up to reveal a square hole in the housing (or slides down, to cover the hole). When the hole is open, the disk is locked. Your computer won't allow you to add anything to the disk.

Hub

The center of the magnetic disk contains a metal hub containing holes. These holes fit over spindles inside the computer and hold the disk in place while it spins.

Paper Rings

The magnetic disk is sandwiched between two white paper rings. The two rings are glued down to the plastic housing and stay still while the disk spins. They clean the disk by removing microscopic bits of dust.

Plastic Flap

Under one of the paper rings is a plastic flap. One end is glued down, and the plastic is bent a little. Like a simple spring, it pushes the paper ring tight against the surface of the magnetic disk.

Working of Floppy Disk

The following six steps explain how these parts of a floppy disk work to record data.

Step 1: Exposing the Recording Surface:

When you insert the floppy disk into the drive, the shutter moves to the side to expose the magnetic recording surface on the disk

Step 2: Sending Signals from the Circuit Board

Next, levers and gears move two read/write heads until they almost touch the magnetic disk on either side. These heads, which are tiny electromagnets, use magnetic pulses to change the orientation of metallic particles embedded in the disk's coating.

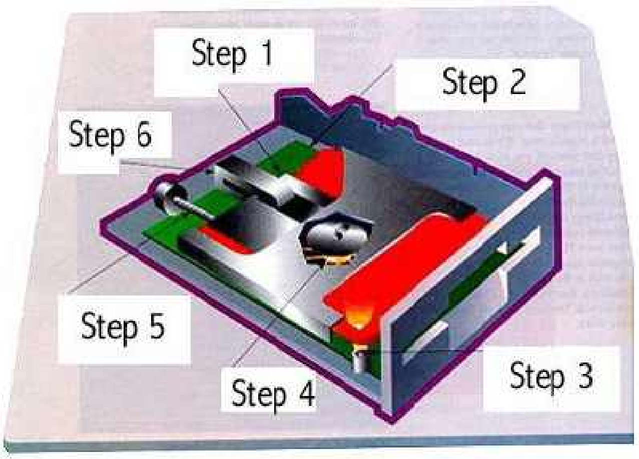


Figure 4-3

The floppy drive's controller board sends signals to the drive's circuit board, including data and instructions for writing data to disk. The circuit board then translates the instructions into signals to control the movement of the disk and the read/write heads.

Step 3: Checking for Write Protection

Next, the circuit board checks if the disk is write protected. If disk access is a write instruction, the circuit board verifies that light is not visible through the write-protect notch. If the notch is open and a beam from a light emitting diode can be detected, the drive knows the disk is write-protected and refuses to record new data.

Step 4: Spinning the Disk

Once the circuit board verifies that data can be written, motor located beneath the disk in the drive spins a shaft. The shaft then engages a notch on the disk's hub, causing the disk to spin.

Step 5: Positioning the Read/Write Heads

Signals from the circuit board then direct a stepper motor, which can turn a specific amount in either direction. This motor positions the read/write heads over the correct location on the recording surface of the disk.

Step 6: Writing the Data

When the heads are in the correct position, electrical impulses create a magnetic field in one of the heads. Data is written to either the top or bottom surface of the disk.

4.2 CD-ROM Compact Disc Read Only Memory

Stands for "Compact Disc Read-Only Memory" A CD-ROM is a CD that can be read by a computer with an optical drive. The "ROM" part of the term means the data on the disc is "read-only," or cannot alter or erased. Because of this feature and their large capacity, CD-ROMs are a great media format for retail software. The first CD-ROMs could hold about 600 MB of data, but now they can hold up to 700 MB. CD-ROMs share the same technology as audio CDs, but they are formatted differently, allowing them to store many types of data.

Since compact discs store audio in a digital format, they are suitable for storing other information that can be represented in a digital form. In 1984, Philips and Sony released the Compact Disc Read Only Memory (CD-ROM) specification, known as the *Yellow Book*. This defines the necessary additions to the Red Book for the storage of computer data. Any computer data can be pre-recorded on a CD-ROM. The specification of a CD-ROM disc is summarized in Table-1

Table 1 CD-ROM Disc Specification

Parameter	**Value**	**Comments**
Data capacity	656 MB	For 74 minutes (688 x 10^6 bytes)
Raw data bitrate	1.41Mbits/s	Includes all bytes in sector
User data rate	150 kB/s	At 1x speed
Block (sector) size	2,352 bytes	
User data per sector	2048	With full error correction
Sector rate	75 sectors/s	At 1x speed reading
Sector Modes	1 or 2	See 3.1
Sector Forms	1 or 2	Mode 2 only

CD-ROM discs are read by CD-ROM drives, which have been standard components of personal computers and some games consoles for a number of years. DVD-ROM drives, which are now replacing CD-ROM drives will also read CD-ROM discs. A CD-ROM has several advantages over other forms of data storage, and a few disadvantages.

The Mode 1 and Mode 2, Form 1 sectors are identical in capacity and error correction. The only difference is the presence of the sub-header in the latter. In contrast Mode 2, Form 2 sectors include no error correction, but the increased capacity (2,324 bytes instead of 2,048) is suitable for video or audio data where occasional errors can be masked and do not cause a problem. Mode 2 Form 2 sectors are used by the Video CD format, for example, for video data and by CD-i for audio, still video and moving video data. For other applications and formats either Mode 1 or Mode 2 Form 1 only are used and the sub-header is not used.

Capacity of a CD-ROM

The capacity of a CD-ROM depends on whether it is a Mode 1 CD-ROM or Mode 2 CD-ROM XA. Assuming the maximum size is 76 minutes 30 seconds (as recommended) this means that there are 336,300 sectors on a CD -ROM. From this must be subtracted 166 sectors at the start of track 1 plus a few sectors for the file system, amounting to, say, 200 sectors leaving 336,100 sectors for user data.

- Mode 1 sectors contain 2048 bytes per sector giving a total capacity of 688,332,800 bytes or 656 MB.
- Mode 2 sectors contain either 2048 or 2324 bytes per sector so will have a somewhat higher data capacity depending on the mix of the two types of sector.

The above assumes a CD-ROM comprising a single track in a single session. For multiple track/session discs the data capacity will be reduced.

Check your progress

Q1. Why floppy disk needed? Explain its components.

Q2. Define CD-ROM. Also explain its working.

4.3 Magnetic Tape Drives

Magnetic tape storage is often the forgotten or ignored peripheral device until the first disk crash or virus infection. In the early days, when disk storage was small and expensive, magnetic tapes were used for data processing as well as storage. A typical process would include reading data in from a set of punched cards. These input records are sorted and written out to tape. Often the sorting process requires the use of two tape drives to perform the sort merge operations as internal memory could be quite limited. The sorted input data tape is mounted on one drive, the "old" master tape on the next drive, and the updated output master on a third drive.

Today the main applications are for archive and backup, where the cost as well as volume per megabyte is still much lower than any other competing medium.

Classification and Application of Tape Drives

There are several ways of classifying magnetic tape storage devices. One way is to classify them by the mode of operation, viz. block mode or streaming mode tape drives. Another way is to divide them according to whether the recording is done by a fixed or a rotating head. The chart in Figure 4-5 gives a finer classification.

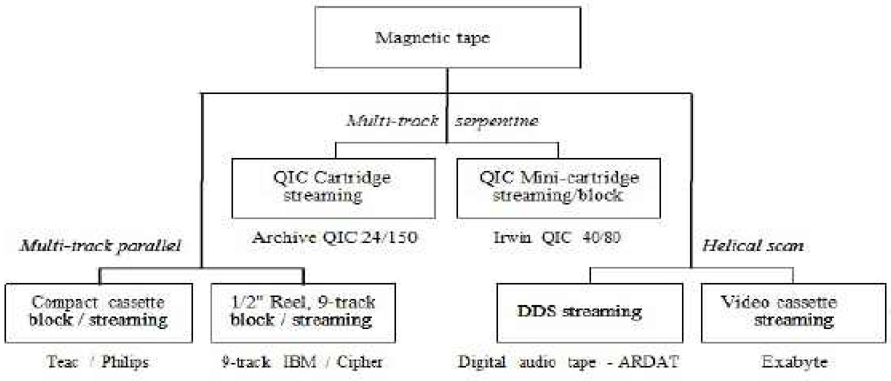


Figure 4-5

Start-Stop Blocked Mode

In the early years of data processing using computers, tape drives were used for the storage of data files on a record-by-record basis. For updating a record or block of data is first read into the CPU, calculations performed, and an updated record is written back to the tape. Provided the block length remains the same, the drive would rewind the tape by one block length and the updated block written back over the old record.

In this blocked mode of operation, data are recorded in fixed physical blocks of 512 to 4096 bytes. Between each block there is an Inter-Record Gap (IRG) of up to 0.5 inch. The magnetic tape controller uses the IRG to locate the beginning of each block and can update or over-write data on the tape on a block by block basis.

Obviously blocked mode operation places a higher demand on the tape transport mechanism. Tape movement now consists a sequence forward, stop, reverse, and stop operations in quick succession. In addition, it has the capability of searching for a specific block by skipping in the forward or reverse directions.

Although blocked mode operation can be implemented on any kind of tape drives, it is usually found only in the large 1/2-inch fixed head multi-track drives and a few mini-cartridge units. In the case of the mini-cartridge units, additional software is used to format the tape into "tracks" and "sectors" so that it emulates a floppy disk drive with a capacity of 20/40 Mbytes.

Table 2 gives the typical specifications of some popular IBM blocked mode 1/2-inch tape drives.

IBM Product No.	726	3420	3480
FCS (First customer shipment)	1953	1973	1985

Linear Density (BPI)	100	6250	38,000
Number of Tracks	7	9	18
Reel Capacity (MB)	2.2	156	200
Data Rate (KBytes/sec)	75	1250	3000
Recording Code	NRZI	GCR(0,2)	GCR(0,3)
Tape Transport	Vacuum	Vacuum	Cartridge

Figure 4-5. Half-inch vacuum column magnetic tape transport shows the mechanism of a vacuum column tape drive. Because of the start-stop operations, keeping the tape well tensioned across the read/write head is non-trivial. Lower cost units, running at low speeds of 45 ips, use spring-loaded arms, but for the higher performance units, a vacuum column is used. The lower pressure within the column draws in excess slack in the tape gently, without causing damage by stretching, or physical contact.

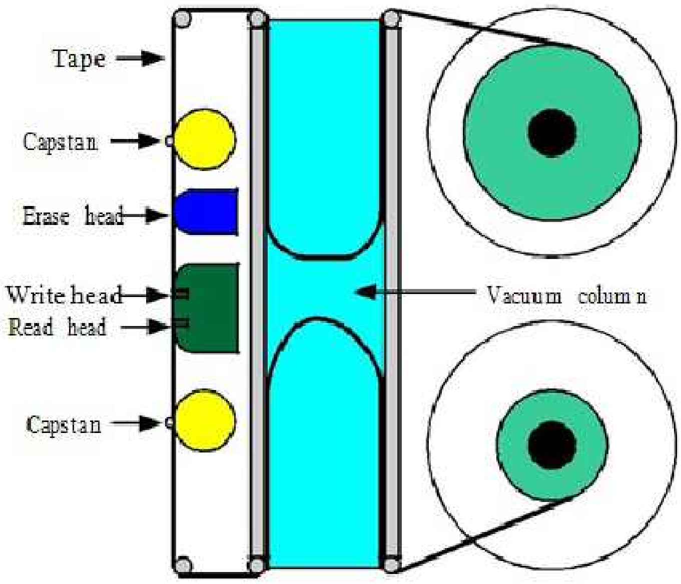


Figure 4-5

Streaming Mode

When the primary application of tape drives is for the archiving and backup, the need to operate in the start-stop mode is no longer required. In such applications, a file or set of files are written onto the tape. If these files have changed, the updated versions are normally kept by re-writing the complete files. No attempt is made to update just the affected records within the file. In this case streaming mode tape drives can be used. Streaming tapes write complete files or groups of files as a continuous "stream" of data. Although the actual data may be recorded in physical blocks, there is no support for locating and modifying a particular block on the tape and leaving the portions before and after unaffected.

Tape motion is controlled only in the forward direction for read, write or skip forward to file mark or end of file. Reverse motion is used only to rewind the tape back to the beginning. We will consider in turn the two current tape drive mechanisms, the fixed head and rotary head machines, both operating in the streaming mode.

Stationary Head Recorders

The ubiquitous Walkman is an audio tape recorder with a stationary head. Early digital tape drives were adaptations of audio tape recorders. In fact normal audio recorders have been used for storing digital data by using the digital information to modulate tone signals in the audio frequency range. FSK, PSK and other modulation schemes have been used. However, digital tape drives normally have different designs to optimize data storage density and transfer. In order to achieve high storage efficiency, direct digital recording is used. The tape is moved rapidly across the head as the data transfer rate is a direct function of tape speed. Digital tape drives run the tape at over 45 ips compared the several inches per second for audio units.

As the recording track runs along the length of the tape, several tracks are laid out in parallel. On the multi-track recorders, these tracks are recorded simultaneously using a head block with a number of recording heads mounted in parallel.

The QIC Standard

The QIC (Quarter-Inch Cartridge) industry standard describes a stationary head, streaming mode tape storage which is becoming increasingly popular because of its compact size and large capacity. However it has only two sets of read/write heads, one for each direction of tape movement. Parallel tracks are obtained by physically moving the head perpendicularly across the width of the tape to access a new track. Each track is written sequentially resulting in a *serpentine* pattern as can be seen in Figure.

QIC defines a number of standards, differing in capacity, as can be seen in Table 2.

	QIC-24	QIC-150	QIC-525	QIC-1350
Capacity (formatted) MB	45 or 60	125 or 150	320 or 525	1.35 GB
Track Format	9	18	26	30
Flux Density	10,000 ftpi	12,500 ftpi	20,000 ftpi	38,750 ftpi
Data Density	8,000 bpi	10,000 bpi	16,000 ftpi	51,667 bpi
Tape Speed	90 ips	90 ips	120 ips	120 ips
Data Transfer Rate	90	112.5	240	600
KBytes/Sec				
Recording Code	GCR (0,2)	GCR (0,2)	GCR (0,2)	RLL(1,7)
Track Width (in)	0.0135	0.0056	0.0070	0.0070
Tape Length (ft)	450 or 600	600	600 or 1000	750
Soft Error Rate	1 in 10^8	1 in 10^8	1 in 10^8	1 in 10^8
Hard Error Rate	1 in 10^{10}	1 in 10^{10}	1 in 10^{10}	1 in 10^{10}

Table-3

QIC-24 Cartridge Tape Drive

We shall use QIC-24 standard to look at the various aspects of the QIC tape drive. Referring again to Figure 4-6 the read/write operation is explained in the following section.

Read/Write Operation

1. When the tape is loaded, rewinding takes place. Tape is then forwarded to Beginning of Tape (BOT), and head is positioned for Track 0.
2. On WRITE command, tape is forwarded to the Load Point (LP). This is the beginning of the recording zone and drive begins writing to tape on Track 0. Data is written bit serial on one track. In addition, during the writing on Track 0, the erase head is enabled. Full track erase is performed by applying AC signal to the erase head ahead of the write head.

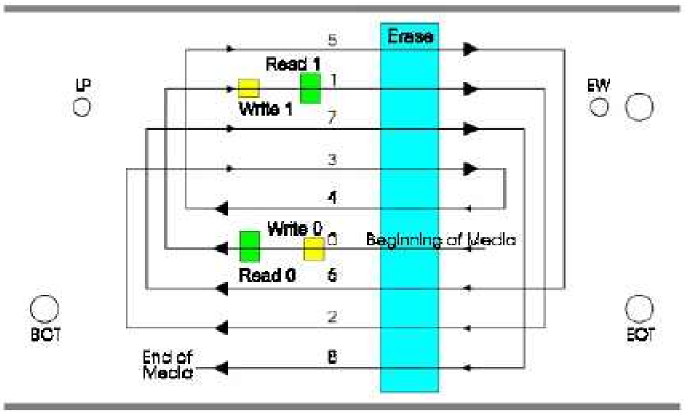


Figure 4-6

3. Writing on Track 0 continues till the Early Warning (EW) hole is detected. Drive stops accepting data from host. Recording stops when all remaining data in buffers are written to tape.
4. Tape movement continues in the forward direction until End of Tape (EOT) is reached. Erase head is disabled. Read / Write head for Track 1 in reverse direction is selected.
5. Tape is now moved in the reverse direction and recording begins when EW hole detected. Recording continues until the LP hole is detected.
6. When end of Track 1 is reached at BOT, tape head is moved (down 0.048 inch from Track 0 to Track 2), and positioned at the next track. Head is moved across width of tape to read/write the various data tracks by attaching it to a screw driven by a stepper motor, with resolution of 0.001 inch.
7. Tape movement is changed back to the forward direction and Track 2 is written.
8. Similar procedures are used, alternating between forward and reverse directions and shifting the head after every pair (F/R) of tracks.
9. The serpentine recording pattern results.
10. The READ Operation is similar.

Data Block Format

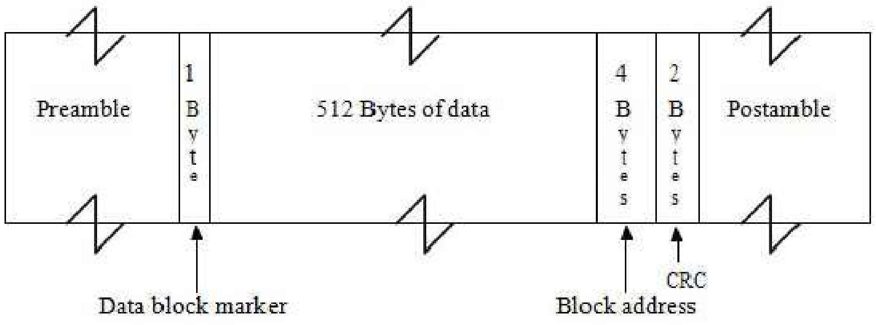


Figure 4-7

Figure 4-7 shows the format of the data block used in the QIC-24 tape drives. The various components comprising the block are described below:

a. **Preamble**

This is a sequence of bytes used to synchronize the PLL to the data frequency. An under run occurs when during the write process, the data rate from the CPU is slower than the write data rate, so that the write buffer in the tape is empty.

Rather than stopping the tape, preamble bytes are written into this area while waiting for the write buffer to fill. A long preamble is used at beginning of each track also.

b. **Data Block Marker GCR** code '11111 00111'

c. **Data Block**

The 512 data bytes encoded as GCR 4/5.

d. **Block Address**

The block address consists of a 4-Byte block address with track number, a control nibble and block number starting with 1 on each track.

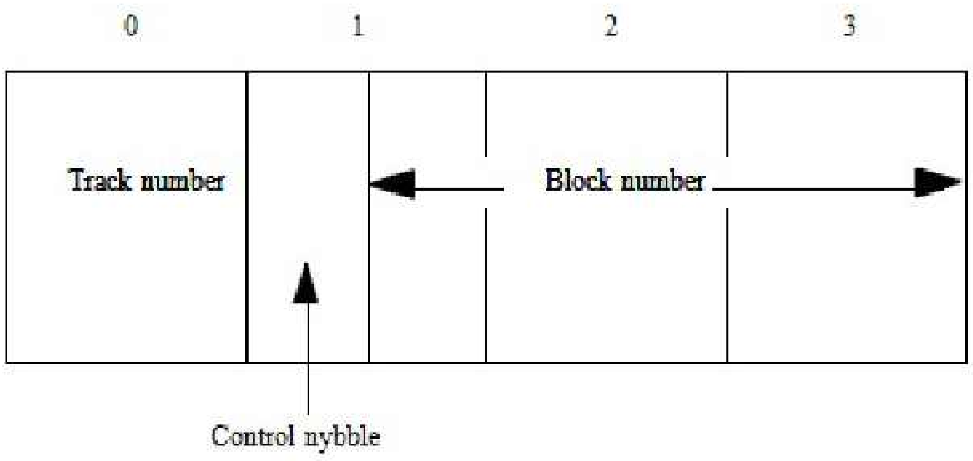


Figure 4-8

e. Cyclic Redundancy Check Character

A 2-Byte CRC character is calculated over the 512 bytes of data and 4 bytes of block address using the following polynomial:

CRC = $x^{16} + x^{12} + x^5 + 1$

f. Post amble

A normal post amble of 5 - 20 flux transitions are recorded at maximum density to act as a guard band. Elongated post ambles of 3500 - 7000 flux transitions are appended when an under run occurs.

Under run

If data from the host are interrupted, or if data transfers from the host drop below 90 Kbytes/sec, under run occurs and duplicate blocks maybe recorded. These duplicate blocks are transparent to the host.

Reliability

The QIC standard has error processing and recovery software incorporated in the firmware. Their operation is transparent to host and user. For each block of 512 bytes of data plus control data, a cyclic redundancy check (CRC) character is computed and recorded. The CRC generating polynomial used is:

$$x^{16} + x^{12} + x^{5} + 1$$

A read check performed on each block of data as it is written. The read head is positioned 0.3 in. as shown in Figure 4-9 after the write head and is used for normal reading and for the read-after-write checking. Based on the tape speed, this distance is equivalent to a delay of approximately 300 bytes.

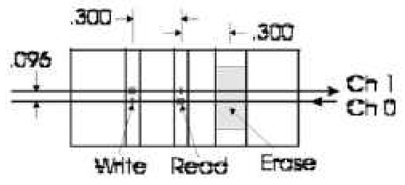


Figure 4-9

Thus when the controller is reading and checking block N, the next block, N+1 is already being written to tape. The following, together with Figure 4-9 describes the protocol used in the error processing and recovery procedure:

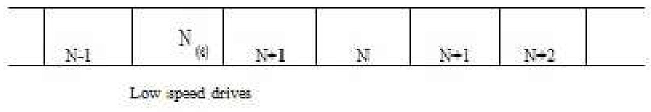
1. Drive begins writing block N (about 530 bytes long).
2. When block N reaches read head (300 bytes later), read checking begins.
3. Writing of block N completes with Block address and CRC.
4. After inter-block gap, block N+1 is written.
5. Read checking finds CRC error in block N.
6. Block N must be re-written, but block N+1 is already half-written.
7. Drive completes writing block N+1 and begins writing block N for second time. A second copy of block N+1 is also written.

8. If there are no errors after second writing of block N during the read check, the tape drive continues with writing block N+1 and the normal sequence resumes.

9. If and error is found at the second write attempt, a third copy of block N (and N+1) is written. This repeated up to a total of 16 attempts.

10. After 16 unsuccessful attempts, the write operation is abandoned.

11. In the READ mode, the last copy of each block is taken as the valid copy.



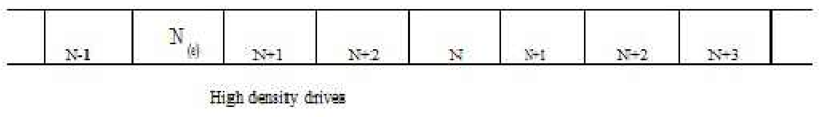


Figure 4-10

QIC-02 Standard Electrical Interface

Op Code	Command	Op Code	Command
01	Select Drive, Soft Lock OFF	80	Read
11	Select Drive, Soft Lock ON	81	Space Forward
21	Position to Beginning of tape	89	Space Reverse
22	Erase Entire Tape	A0	Read File Mark
24	Retension Cartridge	A3	Seek End of Data
40	Write	Bn	Read n File Marks
41	Write without Underrun	C0	Read Status
60	Write file Mark (FM)	C2	Run Self Test 1
80	Read	CA	Run Self Test 2

Table-4

In addition to the various recording capacity/format standards QIC also defined a standard electrical interface and a set of commands, designated QIC-02. These are shown in Table 4 and Figure 4-11 respectively. This means the same tape controller and device driver is usable across the whole range of tape drives whatever the capacity. The only change that may be required is to modify the capacity parameter so that the software can determine when the next volume of tape needs to be mounted.

QIC-36 Standard Drive Interface

A low level electrical interface for the tape drive has also been defined as the QIC-36 standard. This is basically the electrical and electromechanical interface and does not include any intelligence in the tape drive electronics. Essentially this specifies the interface between the read/write head, the motors and sensors on the basic tape drive and the drive controller.

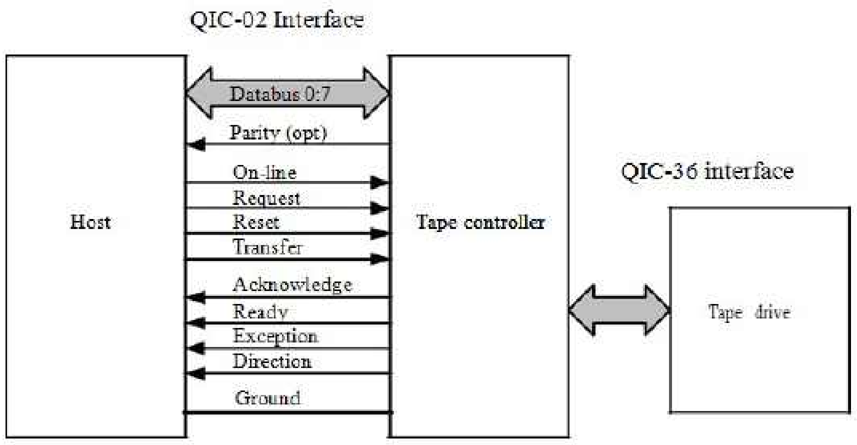


Figure 4-11

Tape Drive Electronics

Figure 4-12 shows the functional block diagram of a typical QIC tape drive with a SCSI interface. The 8-bit microprocessor together with the custom VLSI, provides complete control of the drive. Memory consists of 16 Kbytes of programme in EPROM and 64 Kbytes of dynamic RAM for buffering. The standard 5080 SCSI controller device operates under the control of the microprocessor, negotiating handshake protocols, commands and data transfer with the host computer. The VLSI brings together a number of functions in one reliable package by reducing component count. These are:

- DMA Controller, to handle data transfer from the host through the SCSI controller.
- Interrupt Controller
- Memory Access Controller, provides multiplexing of memory and address buses, dynamic RAM refresh and parity checking.
- Read/Write Controller, in the write mode performs the read-after-write function to verify correct writing of data.
- Clock Generator, provides the various clocks for the microprocessor, SCSI interface and read/write phase clocks.
- Data Separator; when the data is read from tape, it is a mixture of data and clock signals. The read circuit conditions and detects these signals which are then passed to

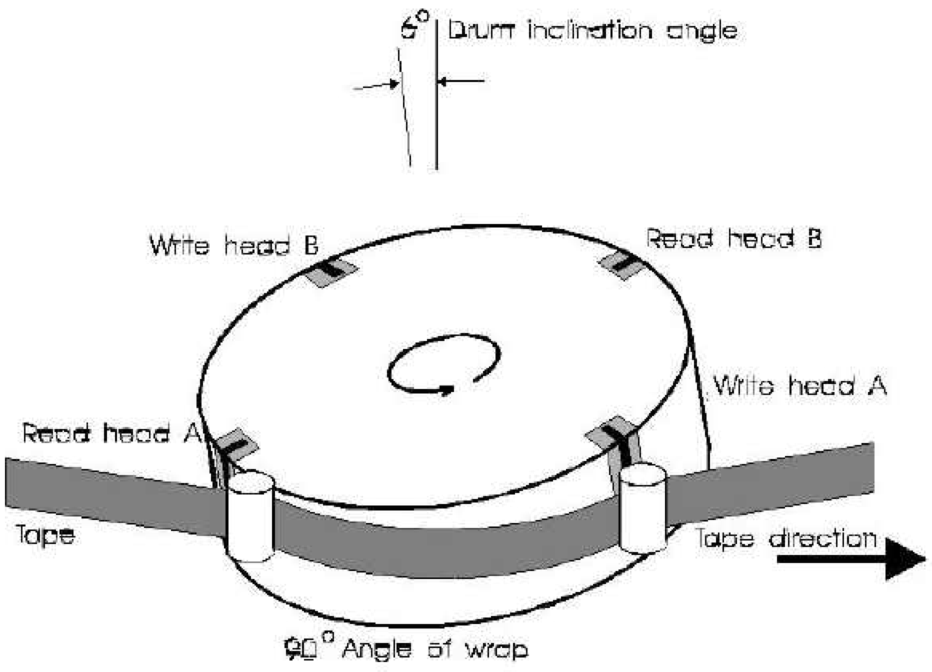


Figure 4-13

Figure 4-13 show the general arrangement of the rotary-head recorder. Four heads are located on the rotating drum, setup as two pairs of write/read heads. Differing from the standard VCR, the tape is only wrapped around a 90°quadrant of the drum, which is rotating at 2000 rpm. The tape itself moves forward at about 1 inch in 3 seconds. It will be seen that each head traces out a diagonal track on the tape. By proper positioning of the read head relative to the associated write head, a read-after write operation can be performed on the same track. The two sets of head and the tape speed are set up in such a way that each set of heads writes adjacent tracks onto the tape. The ANSI Digital Data Storage (DDS) standard format has been proposed for the use of DAT for data storage. A typical specification of a currently available DAT product is given in Table 5.

Product:	Archive Python 4330XT
Capacity:	1.3 GBytes with 60m tape.
Sustained transfer rate:	183 Kbytes/sec, sustained.
Average access time:	20 sec. seek time.
Small form factor:	3 1/2"
Standard recording format:	ANSI DDS

Low cost:	Currently US$0.01 / Mbyte.
Interface:	SCSI-1 and SCSI-2
Media	4 mm. DAT Cartridge, 60/90m length.
Packing density	1869 tracks/in.
Areal density	114 Mbits/sq. in.
Uncorrectable error rate	Using ECC, 1 in 10^{15} bits.
Drum rotation speed:	2000 RPM
Tape speed:	0.32 in/sec.
Search/rewind speed	200 X normal speed
Head-to-tape speed:	123 in/sec, Helical scan (RDAT)

Table -5

DDS DAT Format

DDS DAT is the Digital Data Storage format for Digital Audio Tape, a standard format definition which uses the DAT in a streaming mode. There is a second format defined by Hitachi, Data/DAT, which supports random access. However Data/Dat has a lower data transfer rate and less storage capacity.

Referring to Figure 4-14 it is observed that an area of the tape is allocated for device, reference and vendor information. These are device and vendor specific. There are usually one or two data areas and only one is shown here.

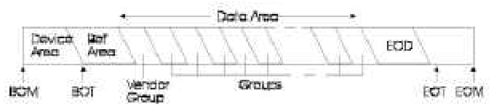


Figure 4-14

Data are organized into groups, with each group containing about 128 Kbytes divided into 22 data frames as shown on Figure 4-15. A DDS DAT device usually has a large buffer so that N groups of data are written at a time.

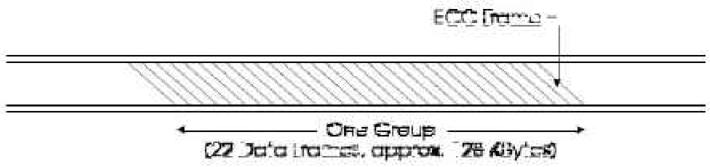


Figure 4-15

Examination of Figure 4-16 shows that each frame is made up of two tracks; track A is written/read by head A and track B by head B respectively. It can be noted that there is no inter-track gap, and each track overlaps the previous track slightly. At the overlap, signals from one track will interfere with the signal from the other. To eliminate this source of cross-talk interference, the heads are given a small azimuth offset of 20°from the normal. In this way, the component of noise from the interfering track is strongly attenuated as there is a total disorientation of 40° between the write-head A and read head B.

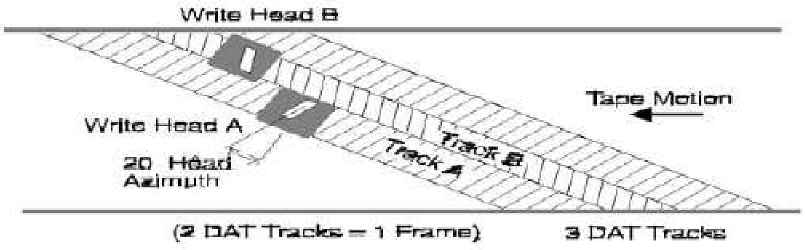


Figure 4-16

Additional processing like interleaving and Reed-Solomon redundancy is used for error correction. Tone bursts are recorded onto the ATF section. On read-back these signals are fed into the track following circuits. In operation, when head B is following track B, the pilot tones from the two adjacent A tracks are sensed and the amplitude difference is use to more accuracy position head B over the track.

Check your progress

Q1. Define Magnetic Tape Drives in detail.
Q2. Explain data block format
Q3. Describe read-write operations.

4.4 Summary

In this unit you learnt about Floppy Disk Drives, Compact Disk Read Only Memory, and Magnetic Tape Drives

- A Floppy Disk Drive is a computer disk drive that enables a user to save data to removable diskettes.
- The shutter and the spring protect the information recorded on the disk.
- The magnetic disk is a round piece of plastic coated with iron oxide, which can be magnetized.
- A CD-ROM is a CD that can be read by a computer with an optical drive.
- Magnetic tape storage is often the forgotten or ignored peripheral device until the first disk crash or virus infection.

4.5 Review Questions

Q1. What is a floppy disk? Why we use floppy disk drive?

Q2. Define all parts of a floppy disk with example.

Q3. What is difference between a Compact disk and a floppy disk?

Q4. Write a note on the Classification and Application of Tape Drives

Q5. Define data block format with example.

BIBLIOGRAPHY

Bradley, A.C. (1989). *Optical storage for computers: technology and applications*. Chichester: Ellis Horwood Limited.

Ceruzzi, P.E. (1998). *A history of modern computing*. Cambridge, M.A.: MIT Press.

Pradeep K. Sinha, Priti Sinha (2004). *Computer fundamentals.* BPB

Larry Long. (2004). *Computer Fundamentals.* New Delhi: Dream Tech Press.

Bissmer, R.H. (1993). *Introduction to Computer Concepts.* New York: John Wiley.

Rajaraman, V (1997). *Fundamentals of Computers. 2nd edition.* New Delhi: Prentice-Hall of India.

Anderson, R.G. (1900). *Data Processing (Vol. 1: Principles and practice; Vol 2: Information Systems and Technology),* London: Pitman.

Ramesh Bangia. *Computer Fundamentals and Information Technology*- FIREWALL Media

Ceruzzi, P.E. (1998). *A history of modern computing.* Cambridge, M.A.: MIT Press.

Bradley, A.C. (1989). *Optical storage for computers: technology and applications.*Chichester: Ellis Horwood Limited.

Jain, Satish (1999). *Information technology.* New Delhi: BPB Publications.

Clemens, A. (2000). The undergraduate curriculum in computer architecture. *IEE Micro*,May /June 2000.

Halal, William E. (1993). The information technology revolution: computer hardware,software and services into the 21st century. *Technological Forecasting and SocialChange*, 44, 69-86.

Hennessy, J.L. and Patterson, D.A. (1990). *Computer architecture: a quantitativeapproach.* N.P: Morgan-Kauffman.

Hussain, Khateeb M. and Hussain, Donna (1989). *Computers: technology applications,and social implications*. New Delhi: Prentice-Hall of India.

Hwang, K. (1993). *Advanced computer architecture*. New York: McGraw-Hill.

Martin, James (1989). *Information engineering: introduction*. New Jersey: Prentice-Hall.

Martin, William J. (1995). *Global information society*. 2nd rev.ed. London: Aslib.pp. 33-63.

Saffady, William. (1992). *Optical storage technology 1992: a state of the art review*.Westport: Meckler.

Vranesic, Z. and Thurber, K. (1980). Teaching computer structures. *Computer*, June1980.

Uttar Pradesh Rajarshi Tandon
Open University

MBA -3.51
Computer Fundamentals and its Organisation

BLOCK

3

Processors & Their Specifications

Unit-I 135
Processor

Unit-II 151
Specifications Of Computer

Unit-III 168
Computer Architecture

Unit-IV 190
Multiprogramming

Course Design Committee

Course Preparation Committee

Printed and Published by Prof. P.P. Dubey, Registrar, Uttar Pradesh Rajarshi Tandon Open University , Prayagraj, 2022.

Printed By : K.C.Printing & Allied Works, Panchwati, Mathura -281003.

Block - 3 : Processors & Their Specifications

This is the third block of this course based on processors & their specifications. This block is very important in terms of processors. It has detailed description about the processors and also focused on processors specifications. This block has four following units.

This is the first unit of this block it has complete description about processor, Basic Processor Structure which has these parts like ALU, Register File, Instruction Register, Control Unit, Clock, Program counter, Memory Address Register, Address Bus, Data Bus, Multiplexed Bus, Executing Instructions. In this unit we also introduce about machine language and instruction set. We also told about the processors used in desktops and laptops.

In the second unit we discussed about the specification of a desktop and laptop computer according to availability in the market. This unit focused on specifications of processor, motherboard & chipset, memory, interface & capacity of hard disk & DVD drives, I/O ports. All these part are very important. With the help of these parts you can assemble a new computer.

In the third unit we focused on the architecture of the computer. It also told about Memory, I/O Module, Processor and data flow between -Memory to processor, Processor to memory, I/O to processor, Processor to I/O, I/O to or from memory. It focused on Bus Interconnection, Bus Structure, Data lines, Address lines, Control lines. This unit also describe about the Bus Hierarchies, Elements of Bus Design.

In the last unit we told about the Interrupt Structures, Multiprogramming, and Processor Features. We also told about the Dual-core Technology and about the Reduced Instruction Set Computers (RISC). This unit also focused on virtual memory, Uses of Virtual Memory, How Virtual Memory Works, Paging, Page Tables, Translation, Translation Look aside Buffer (TLB)

As you study the material you will come across abbreviations in the text, e.g. Sec. 1.1, Eq.(I .I) etc. The abbreviation Sec. stands for section and Eq. for equation. Figure, a. b refers to the bth figure of Unit a, I.e. Figure. 1.1 is the first figure in Unit 1. Similarly, Sec. 1.1 is the first section in Unit 1 and Eq.($.8) is the eighth equation in Unit 4. Similarly Table x. y refers to the yth table of Unit x, I.e. Table. 1.1 is the first table in Unit 1.

In your study you will also find that the units are not of equal-length and your study time for each unit will vary.

We hope you enjoy studying the material and wish you success.

UNIT-I

Processor

Structure

1.0 Introduction

1.1 Structure of Instructions

1.2 Description of a Processor

1.3 Machine Language

1.4 Instruction set

1.5 Processors used in desktops and laptops.

1.6 Summary

1.7 Review Questions

1.0 Introduction

In this unit we focused on the processors of computer. In this unit there are seven sections. In the Sec. 1.1 you will know about structure of instructions. In Sec. 1.2 you will learn about processor. As you know a processor is the logic circuitry that responds to and processes the basic instructions that drive a computer. The term processor has generally replaced the term central processing unit (CPU). The processor in a personal computer or embedded in small devices is often called a microprocessor. In Sec. 1.3 you will learn about machine language. Machine language is the lowest-level programming language (except for computers that utilize programmable microcode). Machine languages are the only languages understood by computers. In Sec. 1.4 we focused on Instruction set. In the next section Sec. 1.5 we describe processors used in desktops and laptops. In Sec. 1.6 and 1.7 you will find summary and review questions respectively.

Objectives

After studying this unit you should be able to:

- Define structure of Instructions, and description of a Processor
- Explain Machine Language and Instruction set
- Express Processors used in desktops and laptops.

1.1 Structure of Instructions

Before we look at basic processor structure, we need to briefly touch on two concepts: von Neumann machines and pipelined, clocked logic systems.

Von Neumann machines

In the early 1950s, John von Neumann proposed the concept of a stored program computer - an architecture which has become the foundation for most commercial processors used today. In a **von Neumann machine**, the program and the data occupy the same memory. The machine has a **program counter** (PC) which points to the current instruction in memory. The PC is updated on every instruction. When there are no branches, program instructions are fetched from sequential memory locations. (A branch simply updates the PC to some other location in the program memory.) Except for a handful of research machines and a very small collection of commercial devices, all of today's commercial processors work on this simple principle.

Synchronous Machines

Again, with a very few exceptions - a handful of research and a small number of commercial systems - most machines nowadays are **synchronous**, that is, they are controlled by a clock.

Datapaths

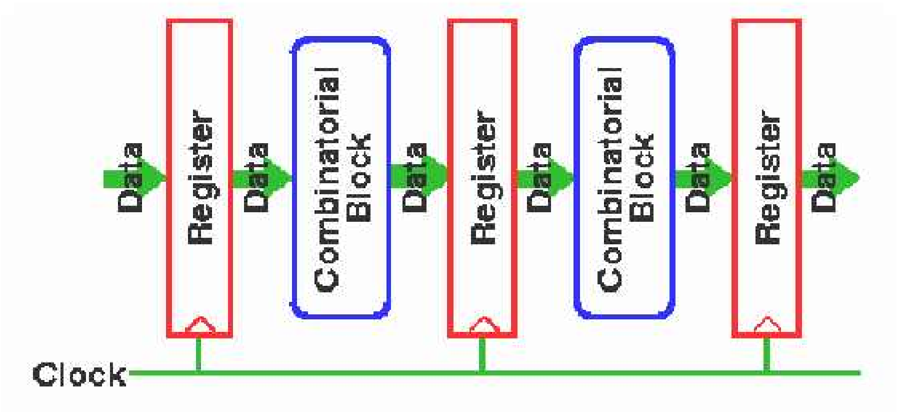


Figure 1-1

Registers and combinatorial logic blocks alternate along the data-paths through the machine. Data advances from one register to the next on each cycle of the global clock: as the clock edge clocks new data into a register, its current output (processed by passing through the combinatorial block) is latched into the next register in the **pipeline.** The registers are master-slave flip-flops which allow the input to be isolated from the output, ensuring a "clean" transfer of the new data into the register. (Some very high performance machines, *eg* DEC's Alpha, use dynamic latches here to reduce propagation delays, *cf* Dobberpuhl *et al.*) In a synchronous machine, the slowest possible propagation delay, t_{pd}^{max}, through any combinatorial block must be less than the smallest clock cycle time, t_{cyc} - otherwise a **pipeline hazard** will occur and data from a previous stage will be clocked into a register again. If $t_{cyc} < t_{pd}$ for any operation in any stage of the pipeline, the clock edge will arrive at the register *before* data has propagated through the combinatorial block.

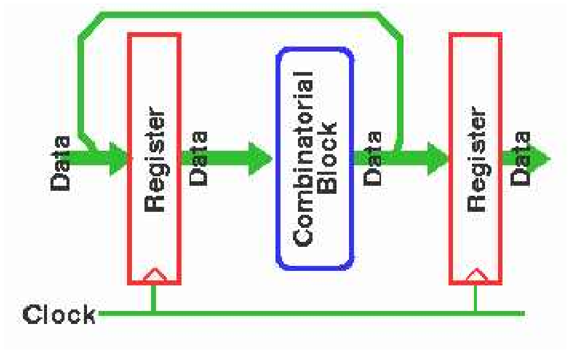


Figure 1-2

Of course, there may also be feedback loops - in which the output of the current stage is fed back and latched in the same register: a conventional state machine.

This sort of logic is used to determine the next operation (*i.e* next microcode word or next address for branching purposes).

Basic Processor Structure

Here we will consider the basic structure of a simple processor. We will examine the flow of data through such a simple processor and identify bottlenecks in order to understand what has guided the design of more complex processors.

Here we see a very simple processor structure - such as might be found in a small 8-bit microprocessor. The various components are:

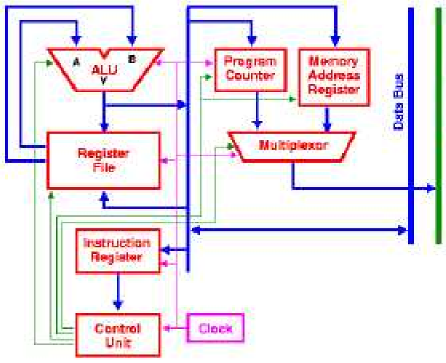


Figure 1-3

ALU

Arithmetic Logic Unit - this circuit takes two operands on the inputs (labeled A and B) and produces a result on the output (labeled Y). The operations will usually include, as a minimum:

- add, subtract
- and, or, not
- shift right, shift left

ALUs in more complex processors will execute many more instructions.

Register File

A set of storage locations (**registers**) for storing temporary results. Early machines had just one register - usually termed an **accumulator**. Modern RISC processors will have at least 32 registers.

Instruction Register

The instruction currently being executed by the processor is stored here.

Control Unit

The control unit decodes the instruction in the instruction register and sets signals which control the operation of most other units of the processor. For example, the **operation code** (opcode) in the instruction will be used to determine the settings of control signals for the ALU which determine which operation (+,-,^,v,~,shift, *etc*) it performs.

Clock

The vast majority of processors are **synchronous**, that is, they use a clock signal to determine when to capture the next data word and perform an operation on it. In a globally synchronous processor, a common clock needs to be routed (connected) to every unit in the processor.

Program counter

The program counter holds the memory address of the next instruction to be executed. It is updated every instruction cycle to point to the next instruction in the program. (Control for the management of branch instructions - which change the program counter by other than a simple increment - has been omitted from this diagram for clarity. Branching instructions and their effect on program execution and efficiency will be examined extensively later.

Memory Address Register

This register is loaded with the address of the next data word to be fetched from or stored into main memory.

Address Bus

This bus is used to transfer addresses to memory and **memory-mapped** peripherals. It is driven by the processor acting as a **bus master.**

Data Bus

This bus carries data to and from the processor, memory and peripherals. It will be driven by the source of data, *ie* processor, memory or peripheral device.

Multiplexed Bus

Of necessity, high performance processors provide separate address and data buses. To limit device pin counts and bus complexity, some simple processors multiplex address and data onto the same bus: naturally this has an adverse effect on performance.

Executing Instructions

Let's examine the steps in the execution of a simple memory fetch instruction, *eg*

$101c_{16}$: lw \$1,0(\$2) This instruction tells the processor to take the address stored in register 2, add 0 to it and load the word found at that address in main memory into register 1.	In this, and most following, examples, we'll use the MIPS instruction set. This is chosen because • it's simple, • it exists in one widely available range of machines produced by SGI and • There is a public domain simulator for MIPS machines, which we will use for some performance studies.
As the next instruction to be executed (our lw instruction) is at memory address $101c_{16}$, the program counter contains 101c.	For convenience, most numbers - especially memory addresses and instruction contents - will be expressed in hexadecimal. When orders of magnitude and performance are being discussed, decimal numbers will be used: this will generally be obvious from the context and the use of exponent notations, *eg* 5×10^{12}.

Table 1-1

Execution Steps

1. The control unit sets the multiplexor to drive the PC onto the address bus.
2. The memory unit responds by placing $8c410000_{16}$ - the lw \$1,0(\$2) instruction as encoded for a MIPS processor - on the data bus from where it is latched into the instruction register.
3. The control unit **decodes** the instruction, recognizes it as a memory load instruction and directs the register file to drive the contents of register 2 onto the A input of the ALU and the value 0 onto the B input. At the same time, it instructs the ALU to add its inputs.
4. The output from the ALU is latched into the MAR. The controller ensures that this value is directed onto the address bus by setting the multiplexor.

5. When the memory responds with the value sought, it is captured on the internal data bus and latched into register 1 of the register file.

6. The program counter is now updated to point to the next instruction and the cycle can start again.

1.2 Description of a Computer Processor

The central processing unit (CPU) is responsible for execution and interpretation of commands from a computer's hardware and software system. It is often called the "brains" of the computer and also acts as a "traffic cop" in routing information in and out of the processor.

Size

- Most CPUs are typically small. This is a stark contrast to just a generation ago when CPUs were units that encompassed an entire room. Today's mainframes consist of smaller CPU units, which can also be found in home computers. The CPU is inserted on the computers' "motherboard."

Arithmetic Logic Unit

- The arithmetic logic unit (ALU) component of the CPU carries out mathematical functions of addition, multiplication, division and subtraction. When a person calculates an item on a spreadsheet, the ALU section of the CPU is used to produce results.

Control Unit

- The control unit regulates the flow of information through the computer according to memory and processor speed. The control unit also tells the data where to go to when the system is calling for a response.

Registers

- Registers hold data to be processed by the CPU. A register consist of an instruction pointer which directs the CPU to the memory location to where information is stored. The information is then routed through the processor.

Check your progress

Q1. Explain Basic Processor Structure.

Q2. Define ALU, CU and Registers.

1.3 Machine Language

```
        169 1 160 0 153 0 128 153 0 129 153 130 153 0 131 200 208 241 96
```

BASIC

```
        5 FOR I=1 TO 1000: PRINT "A";: NEXT I
```

These two programs both print the letter "A" 1000 times on the screen. The ML version takes up 28 bytes of Random Access Memory (RAM). The BASIC version takes up 45 bytes and takes about 30 times as long to finish the job. If you want to see how quickly the ML works, you can POKE those numbers somewhere into RAM and run the ML program with a SYS (Commodore computers) or USR (Atari) or CALL (Apple). In both BASIC and ML, many instructions are followed by an argument. The instructions SYS and CALL have numbers as their arguments. In these cases, the instruction is going to turn control of the computer over to the address given as the argument. There would be an ML program waiting there. To make it easy to see this ML program's speed, we'll load it into memory without yet knowing much about it.

A disassembly is like a BASIC program's LIS Ting. You can give the starting address of an ML program to a disassembler and it will translate the numbers in the computer's memory into a readable series of ML instructions. See Appendix D for a disassembler that you can use to examine and study ML programs.

Program 1-1 64 Versions

Newer model 64's need to have the color registers set before running this program to see the effect on the full screen.

```
1 REM COMMODORE 64 VERSION
800 FOR AD=40000TO40019:READDA:POKE
    AD,DA:NEXTAD
805 PRINT"SYS 40000 TO ACTIVATE"
810 DATA169,1,160,0,153,0
820 DATA4,153,0,5,153,0
830 DATA6,153,0,7,200,208
840 DATA241,96
```

Why Machine Language?

Sooner or later, many programmers find that they want to learn machine language. BASIC is a fine general-purpose tool, but it has its limitations. Machine language (often called assembly language) performs much faster. BASIC is fairly easy to learn, but most beginners do not realize that machine language can also be easy. And, just as learning Italian goes faster if you already know Spanish, if a programmer already knows BASIC, much of this knowledge will make learning machine language easier. There are many similarities.

This block is designed to teach machine language to those who have a working knowledge of BASIC. Following each is a machine language routine which accomplishes the same task. In this way, if you know what you want to do in BASIC, you can find out how to do it in machine language.

To make it easier to write programs in machine language, most programmers use a special program called an assembler. This is where the term "assembly language" comes from. ML and assembly language programs are both essentially the same thing. Using an assembler to create ML programs is far easier than being forced to look up and then POKE each byte into RAM memory. That's the way it used to be done, when there was too little memory in computers to hold languages (like BASIC or Assemblers) at the same time as programs created by those languages. That old style hand-programming was very laborious.

There is an assembler (in BASIC) at the end of this book which will work on most computers which use Microsoft BASIC, including the Apple, PET/CBM, VIC, and the Commodore 64. There is also a separate version for the Atari. It will let you type in ML instructions (like INC 2) and will translate them into the right numbers and POKE them for you wherever in memory you decide you want your ML program. Instructions are like BASIC commands; you build an ML program using the ML "instruction set." A complete table of all the 6502 ML instructions can be found in Appendix A.

It's a little premature, but if you're curious, INC 2 will increase the number in your computer's second memory cell by one. If the number in cell 2 is 15, it will become a 16 after INC 2. Think of it as "increment address two."

Throughout the book we'll be learning how to handle a variety of ML instructions, and the "Simple Assembler" program will be of great help. You might want to familiarize yourself with it. Knowing what it does (and using it to try the examples in this book), you will gradually build your understanding of ML, hexadecimal numbers, and the new possibilities open to the computers who knows ML.

1.4 The Instruction Set

The instruction set, also called instruction set architecture (ISA), is a part of the computer that is related to programming, which is basically machine language. The instruction set provides commands to the processor, to tell it what it needs to do. The instruction set consists of multiple pieces, including addressing modes, instructions, native data types, registers, memory architecture, interrupt, and exception handling, and external I/O.

An example of an instruction set is the x86 instruction set, which is common to find on computers today. Different computer processors can use almost the same instruction set, while still having very different internal design. Both the Intel Pentium and AMD Athlon processors use nearly the same x86 instruction set. An instruction set can be built into the hardware of the processor, or it can be emulated in software, using an interpreter. The hardware design is more efficient and faster for running programs than the emulated software version.

Examples of instruction set

- ADD - Add two numbers together.
- COMPARE - Compare numbers.
- IN - Input information from a device, e.g. keyboard.
- JUMP - Jump to designated RAM address.
- JUMP IF - Conditional statement that jumps to a designated RAM address.
- LOAD - Load information from RAM to the CPU.
- OUT - Output information to device, e.g. monitor.
- STORE - Store information to RAM.

Check your progress

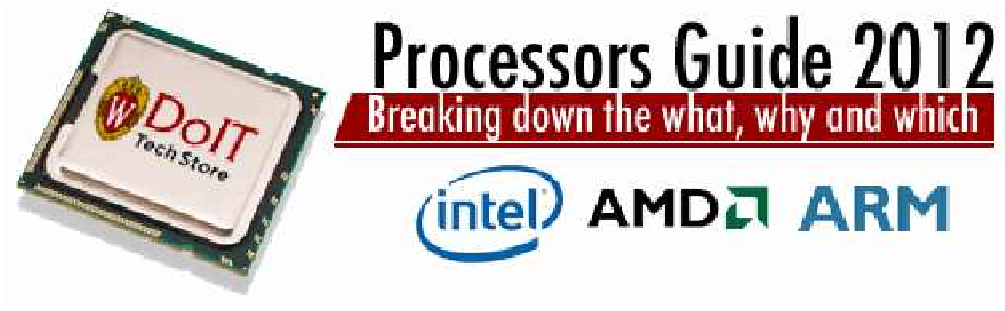
Q1. Why Machine Language needed?

Q2. Define the Instruction Set

1.5 Processors used in desktops and laptops

Intel, AMD & ARM Processors

An overview of notebook and desktop processors offered by Intel and AMD, brief coverage given to ARM whose processors are found in tablets and smartphones.



I. Purpose/Overview

The purpose of this document is to demystify the role that the processor plays in popular consumer electronics, especially laptops and desktop computing systems. Further, this document offers a breakdown of the current state of the processor market–particularly, highlighting the companies, Intel and AMD and how their current product lines size up against each other. Though the emphasis of this document is desktop computing, the spike in relevance of the smartphone and tablet makes mention of mobile processors useful, hence, ARM–a leader in mobile processing is profiled as well. Also provided is a chart that classifies these processors for the purposes of helping you decide on a system appropriate for your needs. Finally, we provide a section dedicated to demystifying some of the technical/marketing jargon that is tossed around as companies introduce new product lines.

II. Why Does the Processor Matter?

The microprocessor–or commonly, the CPU or just processor–is the brain of a computer. It performs many calculations behind the scenes, ultimately allowing you to complete tasks as trivial as composing an e-mail to tasks as intensive as data analysis and modeling. Processors are encountered in many forms of consumer electronics. Most familiar to many are notebook and desktop computers as well as mobile devices such as smartphones and tablets. Though the processor is just one of the many physical components that comprise these products, it is arguably the most central to determining their overall "usefulness" into the future as software requirements become increasingly demanding.

Unlike other components of a notebook computer, the processor is a fixed component. This is in contrast to RAM and hard disk storage which can be upgraded in many cases. Therefore, another consideration is the fact that the CPU you choose will be the same throughout the life of the system. This implies that as applications and operating systems become more sophisticated, the computer's ability to handle them will be directly affected by the purchase decision made all that time ago. This choice may mean the difference between a systems that is useful for another year or two versus one that is not. A final consideration in choosing a CPU is the suggested or minimum requirements of the important software to be used as well as any academic department recommendations as a guide as to the relative kind of computing performance expected for a particular field of study.

III. Companies

(I) Intel

If there was a single semiconductor chip maker the average consumer is aware of it would likely be Intel. If not for the famous Intel "chime" as heard in many television commercials throughout the years then definitely for the fact that it would be difficult not to encounter its technologies in some form whether at work, school or otherwise. Intel is the premier chip maker for personal computers–companies such as Apple, Dell, HP, Samsung, and Sony have product lines that depend on the processors that Intel produces. Intel's processors generally offer the best performance for all-around usage. This has been especially the case the last several years with the introduction and evolution of Intel's Core series product line. Currently, Intel's flagship consumer product line consists of mobile and desktop-grade Core i3, Core i5 and Core i7 processors now in their second generation (dubbed "Sandy Bridge"). The third and latest generation of these processors (dubbed "Ivy Bridge") began to roll out for release late April 2012. The biggest difference between these two generations amounts to a moderate improvement in all-around computing performance but a substantial improvement in integrated graphics performance. Another significant feature Ivy Bridge adds is native USB 3.0 support, overtaking USB 2.0.

(II) AMD

Though not considered the behemoth in the personal computing space as Intel, AMD is a decisive runner-up–and arguably the only true competitor Intel has in this domain. After spending much of the early to middle 2000's as being the performance and value leader with their Athlon 64 line of personal computing processors, AMD–unable to mimic this success in more recent years, has shifted their focus towards both enthusiast and budget-oriented system configurations. As a result, AMD is considered to be a viable alternative to Intel. Their current offerings are flanked by the Phenom series processors and Fusion APU processors. The Fusion APU (AMD A-Series) is a relatively new platform (as of 2011 and ongoing) that attempts to merge high-end graphical capabilities on the same chip as the processor. This means if your work or play requires a powerful graphics card, then AMD can potentially offer a cost effective alternative.

(III) ARM

The increased need for mobile productivity and entertainment has given rise to a relatively new class of devices: smartphones and tablets. ARM is well-known for the design of mobile, power-efficient processor designs. In recent years it has seen its technology used in the products of many prominent electronics companies. Apple's A4/A5/A5X, Nvidia's Tegra, Samsung's Exynos and Texas Instruments' OMAP products all integrate ARM processors into what is known as a system-on-a-chip (SoC). SoCs merge many of the essential components of a computer (such as the CPU, RAM, ROM etc.) on a single chip which allows devices that utilize them to be lightweight and compact. These SoCs have gone on to be implemented in blockbuster products such as Apple's iPhone and iPad or Samsung's series of Galaxy phones. ARM's presence as the CPU and architecture of choice on many mobile devices cannot be understated as estimates put their numbers in the billions.

IV. Benchmarks

It is important to note that there are a considerable amount of details that factor into the overall performance of any given processor beyond frequency (GHz). This is not a valid way to compare most processors, especially between competing companies and between generations. For example, an Intel Pentium 4 (a processor that is generations behind any current Intel i-series processor) running at 3.8 GHz is much slower compared than any i3, i5 or i7 running at lower GHz–the biggest reason for this is the improvement in architecture allowing for increases in the number of cores as well as improvements in other features (such as cache memory and bus technology) that allow the modern CPU to get more work done in a given clock cycle. Hence, it is more useful to compare frequencies and number of cores of processors across the same product line.

CPU benchmarking involves running a specific software tool or suite of tools which allow users to 'distill' an overall performance rating that can be used to rank against other processors.

V. Notable Features Demystified

In this section, we breakdown the practical meaning of some notable technical features included in the various processors available. The vast majority of these features pertain to how a given processor is able to attain a performance boost over either its competitors or previous generations of products.

Feature	**Explanation**	**Processors Using Feature**
Intel Features		
Hyper-Threading (HT)	Improves the performance by allowing the operating system to improve its ability to 'multitask' processes more intelligently. One physically present core is treated as two logical cores which share workloads between each other. Hence, a dual-core with HT has 4 logical cores and a quad-core has 8 logical cores.	Core i7, Core i5, Core i3, Atom
Turbo Boost	Allows the processor to intelligently and dynamically overclock a core(s) such that thermal/power constraints are not violated. For example, a dual core	Core i7, Core i5 (Mobile Dual-Cores only)

	processor with Turbo Boost can overclock one core to much higher frequencies while decreasing speed of the other core; in some situations this can improve performance.	
QuickPath Interconnect (QPI)	An Intel technology which replaced Front Side Bus (FSB) -- similar in purpose to AMD's competing Hyper Transport technology.	Implemented in some fashion across all Intel core iX series
Tri-Gate (3D) Transistor	A new fabrication technology implemented for mass production for the first time in 2012 with Ivy Bridge. Essentially, increases the surface area of each transistor on the chip while also reducing power leakage which on the whole significantly decreases power consumption and improves performance.	Ivy Bridge (2012) iX series
vPro	Synchronizes remote desktop, security, and other multi-station support features. Decreases desk-side maintenance visits.	Current Intel processors
Execute Disable Bit	Prevents certain viruses from infecting the system by labeling some data "executable."	Current Intel processors
AMD Features		
Hyper Transport	A feature that helps minimize the number of buses in a system. This can reduce system 'bottlenecks' and allow microprocessors to use system memory more efficiently.	All current AMD processors

Cool'n' Quiet	Reduces heat and noise of processors allowing for increased energy efficiency.	Phenom I & II, Athlon, Sempron (with exceptions)
Turbo Core	Turbo Core allows for contextual overclocking of the processor to optimize performance subject to electrical and thermal requirements/specifications.	Phenom II X6, Trinity APUs
CoolCore	Limits unused elements of the processor such that power is conserved -- allows for increased notebook battery life on a single charge.	Phenom I & II, Turion

Table -2

Check your progress

Q1. Why Does the Processor Matter? Explain.

Q2. Compare features of Intel and AMD processor.

1.6 Summary

In this unit you learnt about Structure of Instructions, Description of a Processor, Machine Language and Instruction set, Processors used in desktops and laptops.

- Registers and combinatorial logic blocks alternate along the data-paths through the machine.
- ALUs in more complex processors will execute many more instructions.
- A set of storage locations (**registers**) for storing temporary results.
- The control unit decodes the instruction in the instruction register and sets signals which control the operation of most other units of the processor.
- The vast majority of processors are **synchronous**, that is, they use a clock signal to determine when to capture the next data word and perform an operation on it.

1.7 Review Questions

Q1. Define structure of instructions in detail.

Q2. Write a short note on ALU, CU, Registers, Flip flop with example.

Q3. Write a short note on "Description of processors".

Q4. What do you mean by machine language? Why we use it?

Q5. List the name of processors currently used in desktop and laptops.

UNIT-II

Specifications of Computer

Structure

2.0 Introduction

2.1 Specification of a desktop computer currently available in the market

2.2 Specification of a laptop currently available in the market

2.3 Summary

2.4 Review Questions

2.0 Introduction

This unit focused on computer specifications. There are foursections in this unit. In the first section i.e. Sec. 2.1 you will get knowledge about the Specification of a desktop computer currently available in the market. As you know a desktop computer is a personal computer designed for regular use at a single location on or near a desk or table due to its size and power requirements. The most common configuration has a case that houses the power supply, motherboard (a printed circuit board with a microprocessor as the central processing unit, memory, bus, and other electronic component's), disk storage (usually one or more hard disk drives, optical disc drives, and in early models floppy disk drives); a keyboard and mouse for input; and computer monitor and printer for output. The case may be oriented horizontally and placed atop a desk or vertically and placed underneath or beside a desk. An all-in-one desktop computer typically combines the case and monitor in one unit. In Sec. 2.2 you will know about specification of a laptop currently available in the market. As you know a notebook computer is a battery- or AC-powered personal computer generally smaller than a briefcase that can easily be transported and conveniently used in temporary spaces such as on airplanes, in libraries, temporary offices, and at meetings. A notebook computer, sometimes called a laptop computer, typically weighs less than 5 pounds and is 3 inches or less in thickness. Among the best-known makers of notebook and laptop computers are IBM, Apple, Compaq, Dell, Toshiba, and Hewlett-Packard. In Sec. 2.3 and 2.4 you will find summary and review questions respectively.

Objectives

After studying this unit you should be able to:

- Define specification of a desktop computer currently available in the market
- Describe specification of a laptop currently available in the market

2.1 Specification of a desktop computer currently available in the market

Product number

KQ497AA

Release date

13-May-2008

Motherboard

IPIBL-LB (Benicia)

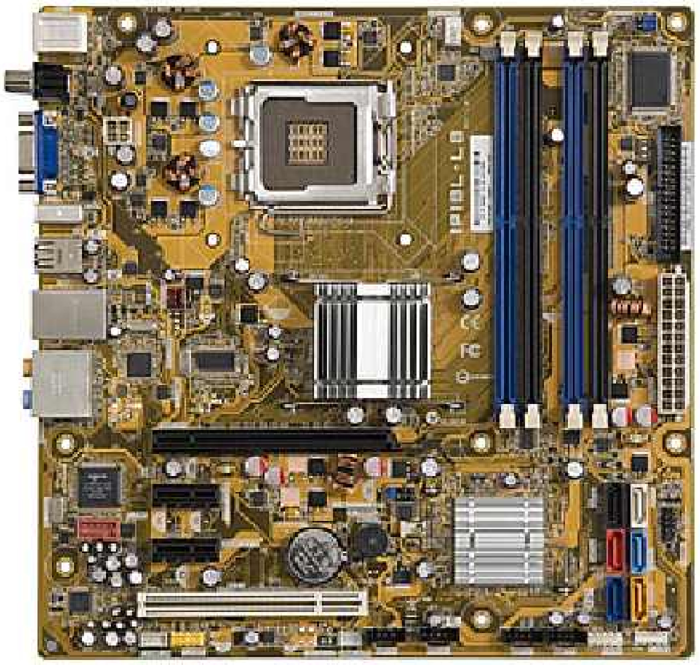


Figure 2-1 Mother Board

- Manufacturer: Asus
- Form factor: micro ATX - 24.4 cm (9.6 inches) x 24.4 cm (9.6 inches)
- Chipset:
- Northbridge chipset: Intel G33 Express

- Southbridge chipset: Intel ICH9R
- Memory sockets: 4 x DDR2
- Front side bus speeds: 1333/1066/800 MHz
- Processor socket: 775
- Expansion Slots:
- 1 PCI x16 slot for graphics card
- 2 PCI x1 slots
- 1 PCI slot

Processor

Intel Core 2 Quad Q6700

Figure 2-2 Processor

- Operating speed: Up to 2.66 GHz
- Number of cores: 4
- Socket: 775
- Bus speed: 1066 MHz

Processor upgrade information

Motherboard supports the following processor upgrades:

- Intel Core 2 Quad (Y) Q9xxxx
- Intel Core 2 Duo (W) E8xxx
- Intel Core 2 Quad (K) up to Q6600
- Core 2 Duo E6x00 (C) up to E6700
- Core 2 Duo E4x00 (C) up to E4400

Memory

6 GB

- Amount: 6 GB
- Speed: PC2-6400 MB/sec

Memory upgrade information

- Dual channel memory architecture
- Four 240-pin DDR2 DIMM sockets
- Supported DIMM types:
- PC2-5300 (667 MHz)
- PC2-6400 (800 MHz)
- Non-ECC memory only, un-buffered
- Supports 2GB DDR2 DIMMs
- Supports up to 8 GB on 64 bit PCs
- Supports up to 4 GB* on 32 bit PCs

*32-bit operating systems cannot address a full 4.0 GB of memory.

Video graphics

NVIDIA GeForce 9500 GS

Figure 2-3 Video Graphics

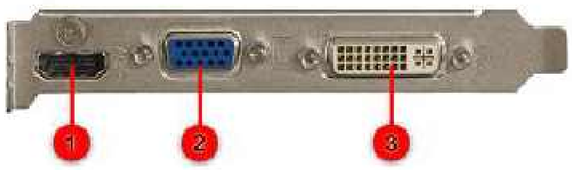


Figure 2-4: Ports

1 - HDMI

2 - VGA

3 - DVI-I (single-link or dual-link)

- Interface: PCI Express x16
- Maximum resolution:
- HDMI TV resolution: up to 1080p*
- HDMI PC resolution: 2560x1600 at 60 Hz
- DVI TV resolution: up to 1080p*
- DVI PC resolution (dual-link): 2560x1600 at 60 Hz
- DVI PC resolution (single-link): 1920x1200 at 60 Hz
- VGA resolution: 2048x1536 at 85 Hz

** Use NVIDIA software to optimize display*

- 512 MB onboard memory
- Supports two displays at the same time*

** Two of the three ports are usable at any one time*

- Supports HDCP

Integrated graphics using Intel GMA 3100

*Integrated video is not available if a graphics card is installed

- Integrated graphics using Intel GMA 3100
- Also supports PCI Express x16 graphics cards*

Sound/Audio

Integrated Audio ALC888S

*Integrated audio is not available if a sound card is installed.

- Audio CODEC: ALC888S
- 7.1 channel high-definition audio

TV-Tuner

Hauppauge NTSC and ATSC

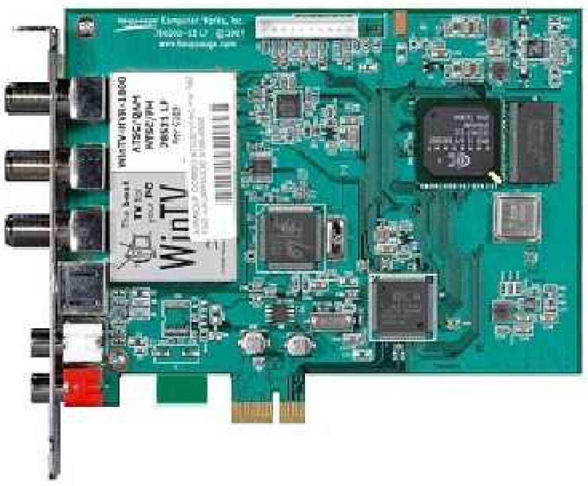


Figure 2-5: TV Tuner card top view

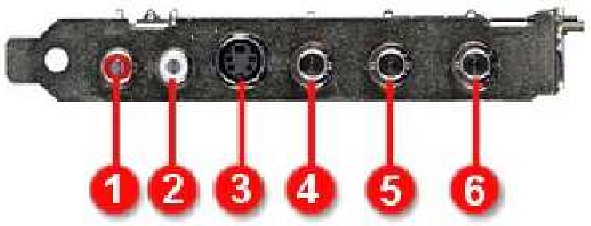


Figure 2-6: Ports

1 - Right audio in (R)

2 - Left audio in (L)

3 - S-Video in (S-Vid)

4 - Coaxial ATSC (ANT IN)

5 - Coaxial NTSC (TV In)

6 - Coaxial FM radio

- Interface: PCI Express x1
- Supported transmission standards:
- ATSC (digital)*
- NTSC (analog)
- FM Radio*

** Requires an antenna*

- Both ATSC and NTSC connections can be used at the same time.
- A 10-pin header on the card provides additional S-Video, composite video, and audio inputs for multimedia connections on the front of some PC cases.

Networking

LAN: 10-Base-T

- Interface: Integrated into motherboard
- Technology: Realtek 8111C
- Data transfer speeds: up to 10/100 Mb/s
- Transmission standards:10-Base-T Ethernet

802.11 wireless b/g/n PCI-Express x1 card

Figure 2-7: Wireless card - side view

- Interface type: PCI
- Data transfer speeds: up to 300 Mbps
- Transmission standards: 802.11 b/g/n
- Supported antenna connections: Either 2 internal antennas or 1 external antenna
- Operating band: 2.4 GHz to 2.5 GHz
- Supported security protocols:
- WPA
- WEP

Modem: 56K WinModem

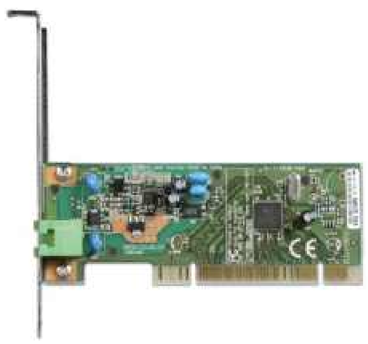


Figure 2-8: Modem card side view

- Interface: PCI 2.2
- Data transfer speeds: up to 56 kbps for V.92 connections*
- Transmission standards: V.92, V.90, V.44, V.34*

** Features must supported by the ISP at time of connection.*

Hard drive

750 GB

Figure 2-9: Hard drive

- Size: 750 GB
- Interface: SATA
- Transfer rating: 3.0 Gb/sec
- Rotational Speed: 7200 RPM

CD/DVD disc drive

Super Multi 16X DVD(+/-)R/RW 12X RAM (+/-)R DL Light Scribe SATA drive

Figure 2-10: Optical Drive

- Interface: SATA
- Data buffer memory: 2 MB
- Maximum storage capacity: 8.5 GB
- Must use Double-Layer media discs in order to take advantage of the DL technology
- Must use Light Scribe-enabled media discs and supporting software in order to take advantage of the Light Scribe technology

Table2-1

Function	Maximum speed
DVD-RAM	up to 12x
DVD+R DL write once	up to 8x
DVD-R DL write once	up to 8x
DVD+R write once	up to 16x
DVD+RW rewriteable	up to 8x
DVD-R write once	up to 16x
DVD-RW rewriteable	up to 6x
DVD-ROM read	up to 16x
CDR write once	up to 40x
CDRW rewriteable	up to 32x
CDROM read	up to 40x

Memory Card Reader

15-in-1 multimedia card reader with infrared receiver

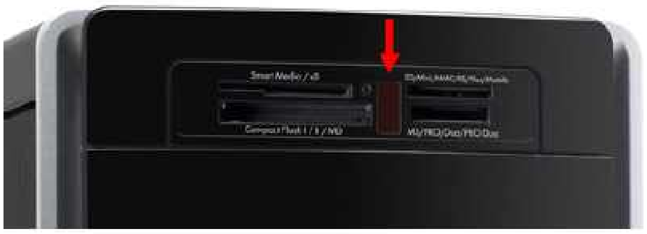


Figure 2-11: View of memory card reader

- IR Receiver
- Supports the following cards:
- CompactFlash I
- CompactFlash II
- IBM Microdrive
- Secure Digital (SD)

- mini-SD
- Multi Media Card (MMC)
- Reduced size Multi Media Card (RS-MMC)
- Multi Media Card Plus (MMC plus)
- Multi Media Card Mobile (MMC mobile)
- Memory Stick
- Memory Stick Pro
- Memory Stick Duo
- Memory Stick Pro Duo
- Smart Media
- xD-Picture Card (xD = extreme digital)

I/O Ports

Front I/O ports

- 1394: 1
- USB: 2
- Headphone: 1
- Microphone: 1
- Audio L-R: 1
- S-video in: 1
- Composite video in: 1

Back I/O ports

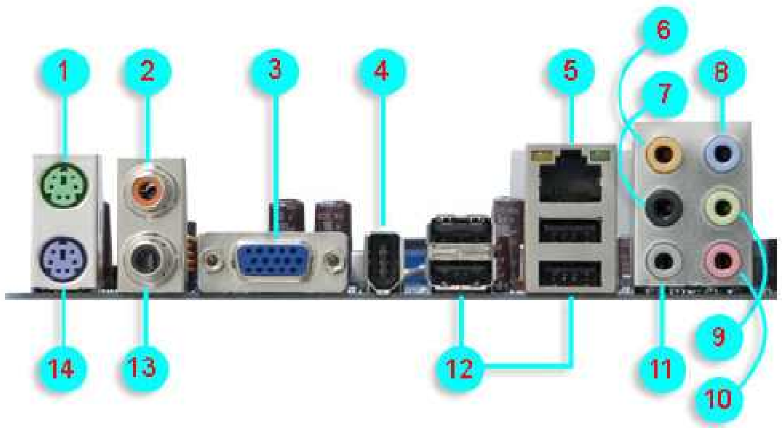


Figure 2-12: Back I/O panel

1 - PS/2 mouse port (green)

2 - Coaxial SPDIF Out

3 - Video Graphics Adapter

4 - IEEE 1394a

5 - RJ45 Network (LAN)

6 - Audio: Center/Subwoofer (yellow orange)

7 - Audio: Rear Speaker Out (black)

8 - Audio: Line In (light blue)

9 - Audio: Line put (lime)

10 - Audio: Microphone (pink)

11 - Audio: Side Speaker Out (gray)

12 - USB 2.0 ports 1, 2, 3, and 4

13 - Coaxial SPDIF In

14 - PS/2 keyboard (purple)

Keyboard, mouse and input devices

HP multimedia keyboard

Figure 2-13: Top view of keyboard

- Interface Type: PS/2
- Cable length: 2 meters (78.74 inches)
- Special buttons:

HP PS/2 optical mouse

Figure 2-14: Top view of mouse

- Interface: PS/2
- Buttons: left (select), right (context menu), scroll wheel (scroll up/scroll down)

2.2 Laptop Specification available in the market

System Features

Operating system

Windows 8 64
Windows 8 Pro 64
Windows 7 Professional (available through downgrade rights from Windows 8 Pro)
Windows® 7 Professional 32
Windows® 7 Professional 64
Windows® 7 Home Premium 64
Free DOS

Processor family

- Intel® Core™ i7 processor
- Intel® Core™ i5 processor

Processors

- Intel® Core™ i5-3320M (2.6 GHz, 3 MB cache, 2 cores)
- Intel® Core™ i5-3340M (2.7 GHz, 3 MB cache, 2 cores)
- Intel® Core™ i5-3360M (2.8 GHz, 3 MB cache, 2 cores)
- Intel® Core™ i5-3380M (2.9 GHz, 3 MB cache, 2 cores)
- Intel® Core™ i7-3520M (2.9 GHz, 4 MB cache, 2 cores)
- Intel® Core™ i7-3540M (3 GHz, 4 MB cache, 2 cores)
- Intel® Core™ i7-3610QM (2.3 GHz, 6 MB cache, 4 cores)
- Intel® Core™ i7-3630QM (2.4 GHz, 6 MB cache, 4 cores)
- Intel® Core™ i7-3720QM (2.6 GHz, 6 MB cache, 4 cores)
- Intel® Core™ i7-3740QM (2.7 GHz, 6 MB cache, 4 cores)
- Intel® Core™ i7-3820QM (2.7 GHz, 8 MB cache, 4 cores)
- Intel® Core™ i7-3840QM (2.8 GHz, 8 MB cache, 4 cores)

Processor technology

Intel® Core™ i5 with vPro technology (select models); Intel® Core™ i7 with vPro technology (select models)

Chipset

Mobile Intel® QM77 Express

Environmental

Low halogen

Dimensions and weight

Dimensions (W x D x H)

3.18 x 33.8 x 23.13 cm

Weight

Starting at 2.25 kg
(Starting weight is with SSD and weight saver (no optical drive))

Memory

Memory, maximum

16 GB 1600 MHz DDR3 SDRAM

Memory slots

2 SODIMM

Storage

Internal drive

320 GB up to 750 GB SATA II (7200 rpm)
500 GB SED SATA II (7200 rpm)
128 GB up to 180 GB Solid State Drive
256 GB SED Solid State Drive

Optical drive

- Blu-ray R/RE DVD+/-RW SuperMulti DL
- Blu-ray ROM DVD+/-RW SuperMulti DL
- DVD+/-RW SuperMulti DL
- DVD-ROM

(*An optical drive, 500 GB 7200 rpm hard drive, or weight saver may populate the upgrade bay.*)

Display and graphics

Display

14" diagonal LED-backlit HD anti-glare (1366 x 768); 14" diagonal LED-backlit HD+ anti-glare (1600 x 900)

Graphics

- AMD FirePro™ M2000 (1 GB dedicated GDDR5)

Expansion Features

Ports

- 2 USB 3.0
- 1 USB 2.0 charging
- 1 eSATA/USB 2.0 combo
- 1 DisplayPort
- 1 1394a
- 1 stereo microphone In
- 1 stereo headphone/line-out
- 1 AC power
- 1 RJ-11
- 1 RJ-45
- 1 docking connector
- 1 secondary battery connector
- 1 VGA

Expansion slots

- 1 Express Card/54
- 1 Secure Digital
- 1 Smart Card Reader

Audio

- HD Audio
- SRS Premium Sound PRO (Windows OS only)
- Integrated stereo speakers
- Integrated microphone (dual-microphone array when equipped with optional webcam)
- Button for volume mute
- function keys for volume up and down
- Stereo headphone/line out

- Stereo microphone in

Webcam

720p HD webcam (select models)

Keyboard

Spill-resistant keyboard and drain

Pointing device

- Touchpad with on/off button, two-way scroll, gestures, two pick buttons
- Pointstick with two additional pick buttons

Communications

Network interface

Integrated Intel 82579LM Gigabit Network Connection (10/100/1000) (vPro configurations)

Wireless

- HP hs2350 HSPA+ Mobile Broadband
- HP un2430 EV-DO/HSPA Mobile Broadband
- Intel Centrino 802.11a/b/g/n
- Broadcom 802.11a/b/g/n
- HP Integrated Module with Bluetooth 4.0+ EDR

Power and operating requirements

Energy efficiency

- ENERGY STAR® qualified configurations available

Power supply

90W Smart AC adapter; HP Fast Charge (not supported on 9-cell primary battery)

Battery type

- 9-cell (100 WHr) Li-Ion battery
- 6-cell (62 WHr) Li-Ion battery
- HP Long Life 6-cell (55 WHr) Li-Ion battery

Battery life

6-cell (62 WHr) Li-Ion battery: Up to 5 hours and 45 minutes

Security management

- Standard: HP Client Security Dashboard (Windows 8 only)
- HP ProtectTools Security Manager includes: Credential Manager, Device Access Manager, Drive Encryption, Face Recognition (webcam configurations only), File Sanitizer (Windows 7 only), Privacy Manager (Windows 7 only), Embedded Security (Windows 7 only)
- HP Fingerprint Sensor
- Integrated Smart Card Reader
- Enhanced Pre-Boot Security
- HP Spare Key
- One-Step Logon
- TPM 1.2 Embedded Security Chip
- Security lock slot
- Support for Intel AT (requires Absolute Computrace subscription) (select models)
- Optional: HP Privacy Filter

Check your progress

Q1. Define specification of a desktop computer currently available in the market.

Q2. Explain laptop specification available in the market.

2.3 Summary

In this unit you learnt about basics knowledge of specifications of Specification of a desktop and Laptop computer currently available in the market. These Specifications is based on Processor, motherboard & chipset, memory, interface & capacity of hard disk & DVD drives, I/O ports

- Desktop processor's Operating speed is available in the market Up to 2.66 GHz
- Desktop Mother board is Core 2 Duo E6x00 (C) up to E6700
- Desktop Memory Speed: PC2-6400 MB/sec and size is 6GB
- In the laptop Operating System 8 is available
- Memory of laptop is 6 GB

2.4 Review Questions

Q1. What is the role of motherboard in the desktop computers?

Q2. What is the minimum requirement of windows 8 operating system?

Q3. What is the difference between desktops hard disk and laptops hard disk?

UNIT-III

Computer Architecture

Structure

3.0 Introduction

3.1 Interconnection of Units

3.2 Processor to Memory communication

3.3 I/O to Processor Communication

3.4 Summary

3.5 Review Questions

3.0 Introduction

In this unit we focused on computer architecture. There are five sections in this unit. In section 3.1 you will know about interconnection of units. A modern computer system is an interconnection of one or more processors, memory units, and input/output (I/O) devices. A personal computer (i.e., a microcomputer) additionally may include an optional special-purpose or custom processor used as an accelerator (e.g., GPU, FPGA). A keyboard, mouse, printer, network adapter, hard disk or flash drive, portable drive (e.g., memory stick), CD drive, microphone, etc. are examples of I/O devices used in a microcomputer. In Sec. 3.2you will learn about processor to memory communication. Communication between memory and processing unit consists of two registers: Memory Address Register (MAR) and Memory Data Register (MDR). These register can be used to read-The address of the location is put in MAR, The memory is enabled for a read and the value is put in MDR by the memory. These registers also used to write- the address of the location is put in MAR, the data is put in MDR, the Write Enable signal is asserted and the value in MDR is written to the location specified. In Sec. 3.3 we focused on I/O to Processor Communication. In Sec. 3.4 and 3.5 you will find summary and review questions respectively.

Objectives

After studying this unit you should be able to:

- Define Interconnection of Units
- Express processor to Memory communication
- Describe I/O to Processor Communication

3.1 Interconnections Structures

A computer consists of a set of components or modules (processor, memory, I/O) that communicate with each other. A computer is a network of modules. There must be paths for connecting these modules. The collection of paths connecting the various modules is called the **Interconnection structure.**

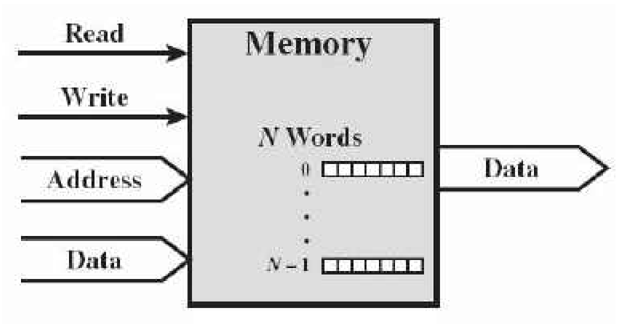


Figure 3-1

- **Memory**
    - Consists of N words of equal length
    - Each word assigned a unique numerical address (0, 1, ..., N-1) o A word of data can be read or written
    - Operation specified by control signals.
    - Location specified by address signals

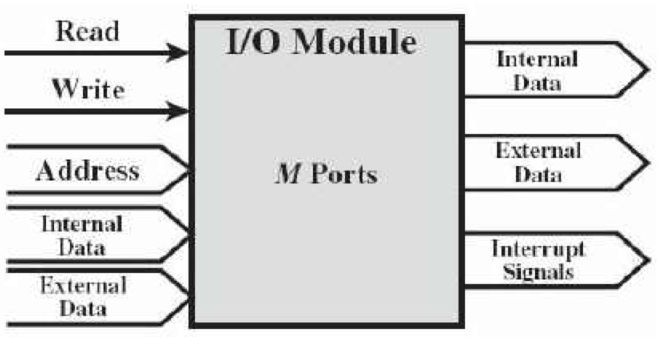


Figure 3-2

• **I/O Module**

- Similar to memory from computers viewpoint
- Consists of M external device ports (0, 1, ..., M-1) o External data paths for input and output
- Sends interrupt signal to the processor

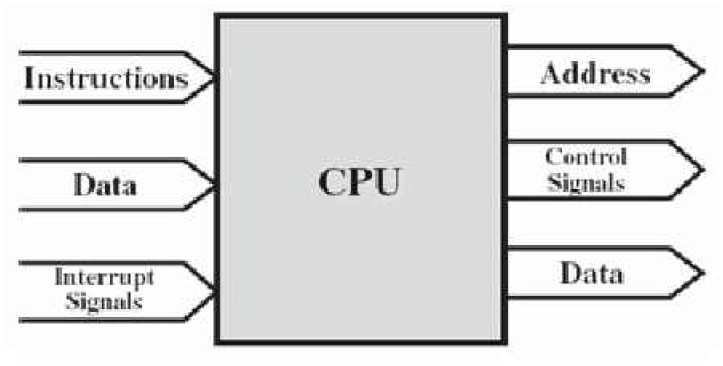


Figure 3-3

• **Processor**

- Reads in instructions and data o Writes out data after processing
- Uses control signals to control overall operation of the system o Receives interrupt signals

The preceding list defines the data to be exchanged. The interconnection structure must support the following types of transfers:

- **Memory to processor**: processor reads an instruction or a unit of data frommemory.
- **Processor to memory**: processor writes a unit of data to memory.
- **I/O to processor**: processor reads data from an I/O device via an I/O module.
- **Processor to I/O**: processor sends data to the I/O device via an I/O module.
- **I/O to or from memory**: an I/O module is allowed to exchange data directly with memory, without going through the processor, using direct memory access (DMA).

Over the years, a number of interconnection structures have been tried. By far the most common is the bus and various multiple-bus structures.

Bus Interconnection

A bus is a communication pathway connecting two or more devices. Multiple devices can be connected to the same bus at the same time. Typically, a bus consists of multiple communication pathways, or lines. Each line is capable of transmitting signals representing binary 1 or binary 0. A bus that connects major computer components (processor, memory, I/O) is called a system bus.

Bus Structure

Typically, a bus consists of 50 to hundreds of separate lines. On any bus the lines are grouped into three main function groups: data, address, and control. There may also be power distribution lines for attached modules.

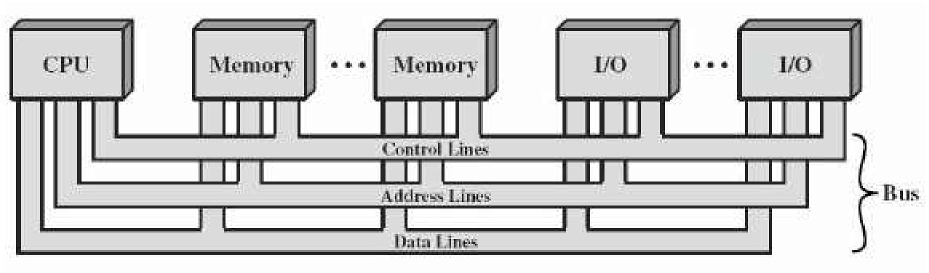


Figure 3-4

Data lines

- Path for moving data and instructions between modules.
- Collectively are called the data bus.
- Consists of: 8, 16, 32, 64, etc... bits - key factor in overall system performance

Address lines

- Identifies the source or destination of the data on the data bus.
- CPU needs to read an instruction or data from a given memory location.
- Bus width determines the maximum possible memory capacity for the system.
- 8080 has 16 bit addresses giving access to 64K address

Control lines

- Used to control the access to and the use of the data and address lines.
- Transmits command and timing information between modules.

Typical control lines include the following:

- Memory write: causes data on the bus to be written to the addressed memory location.
- Memory read: causes data from the addressed memory location to be placed on the bus.
- I/O write: causes data on the bus to be output to the addressed I/O port.
- I/O read: causes data from the addressed I/O port to be placed on the bus.
- Transfer ACK: indicates that data have been from or placed on the bus.
- Bus request: indicates that a module needs to gain control of the bus.
- Bus grant: indicates that a requesting module has been granted control of the bus.
- Interrupt request: indicates that an interrupt is pending.
- Interrupt ACK: indicates that the pending interrupt has been recognized.
- Clock: used to synchronize operations.
- Reset: initializes all modules.

What does a bus look like?

- Parallel lines on a circuit board.
- Ribbon cables.
- Strip connectors of a circuit board.
    - PCI, AGP, PCI Express, SCSI, etc..

Bus Hierarchies

If a great number of devices are connected to the bus, performance will suffer.

- Propagation delays

- Long data paths mean that coordination of the bus use can adversely affect performance.

- If aggregate data transfer approaches capacity
    - Increasing data rate or making the bus wider may help.

Most computer systems use multiple buses, generally laid out in a hierarchy.

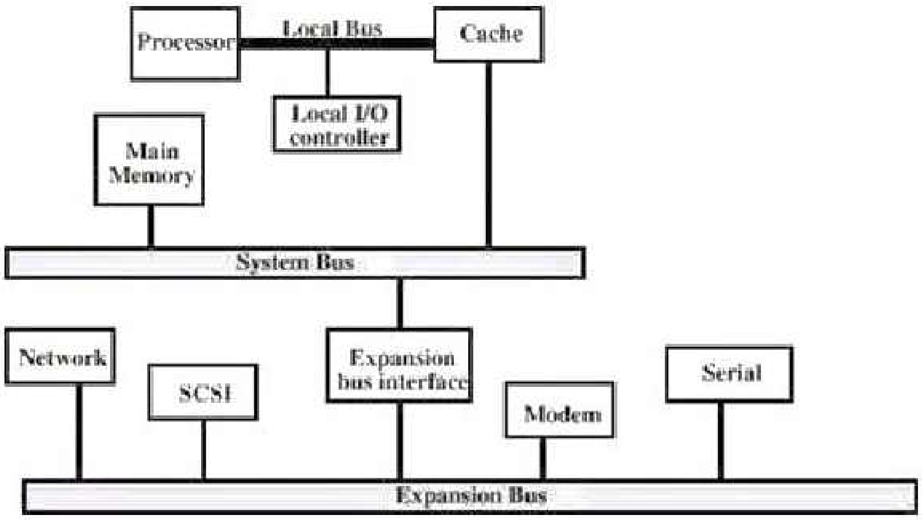


Figure 3-5

The traditional bus architecture shown in the previous figure is reasonably efficient but begins to break down as higher and higher performance is seen in the I/O devices. In response to these growing demands, a common approach taken by industry is to build a high-speed bus that is closely integrated with the rest of the system, requiring only a bridge between the processor's bus and the high-speed bus. This arrangement is sometimes known as mezzanine architecture. The figure below shows an example of using this approach:

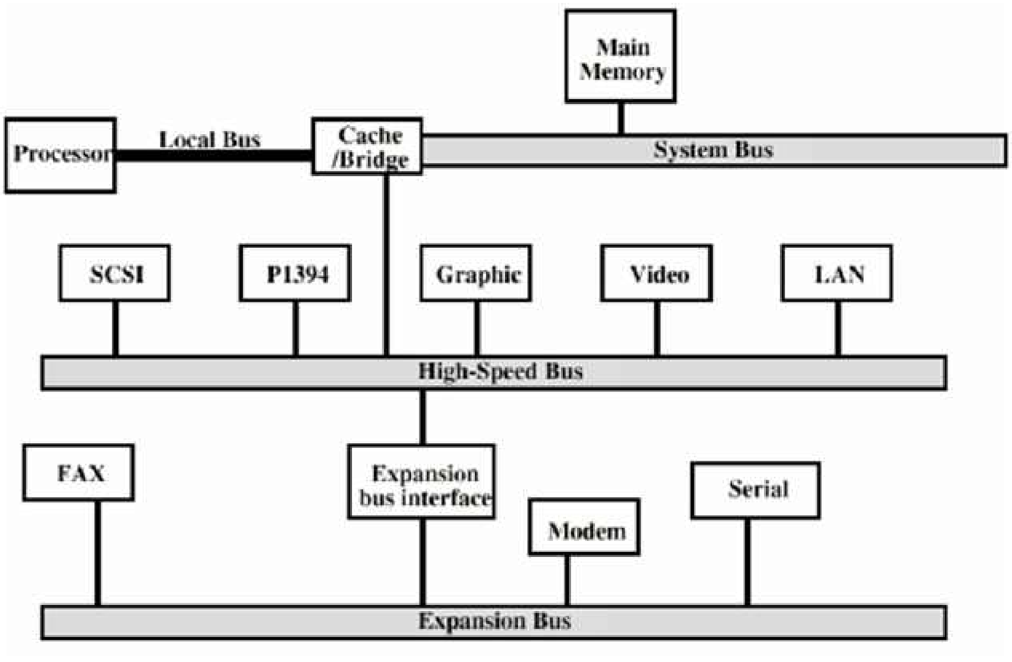


Figure 3-6

Elements of Bus Design

Bus Type

- Dedicated
  - Separate data and address lines
- Multiplexed
  - Shared lines
  - o Address valid or data valid control line

Advantage

 Fewer lines - saves space, lower cost

Disadvantage

 More complex control

 Potential reduction in performance

Method of Arbitration

Because only one unit at a time can successfully transmit over the bus, some method of arbitration is needed.

- Centralized
  - Bus controller of arbiter is responsible for allocating time on the bus
  - May be part of processor or may be separate
- Distributed
  - Each module may claim control of the bus
  - Each module contains access control logic

Timing

Timing refers to the way in which events are coordinated of the bus.

- Synchronous
    - Events determined by clock signals
    - Control bus includes a clock line
    - A single 1-0 is a bus cycle
    - Usually synched on leading edge
    - Usually a single cycle for an event
- Asynchronous
    - One event on a bus follows and depends on the occurrence of a previous event

Synchronous timing is simpler to implement and test. However, it is less flexible that asynchronous timing.

Bus width

The width of the data bus has an impact on system performance: The wider the data bus, the greater the number of bits transferred at one time.

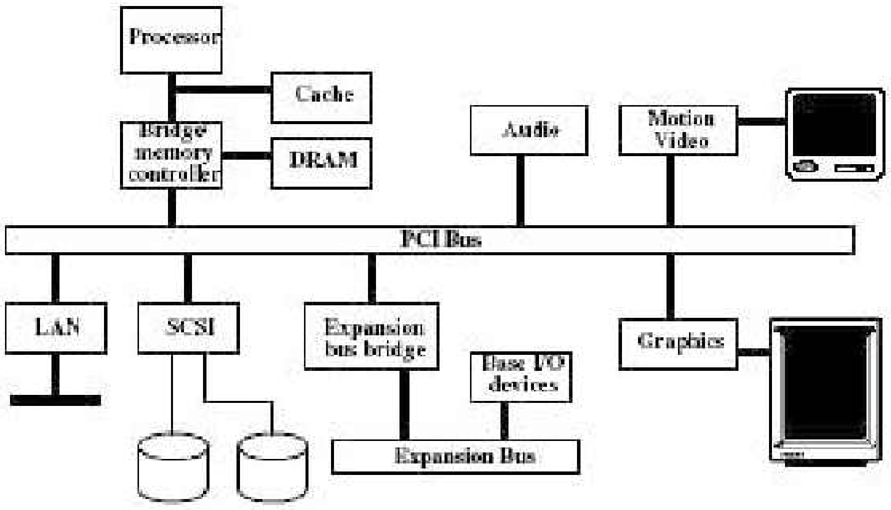
The width of the address bus has an impact on system capacity: The wider the address bus, the greater the range of locations that can be referenced.

Data Transfer Type

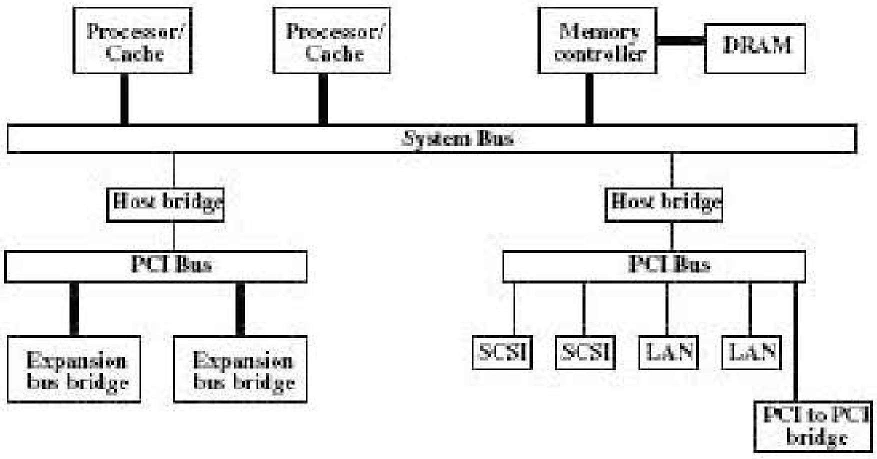
- Multiplexed
    - Single shared bus for both addresses and data
    - Bus first used to specify an address
    - Bus then used to transfer data
- Dedicated
    - Separate buses for both address and data
    - The address is specified on the address bus and remains while data transferred
    - The data is transferred on the data bus

PCI

- Peripheral Component Interconnection
    - 1990 Intel released to public domain 66 MHz 528 Mbytes/sec



(a) Typical Desktop System



(b) Typical Server System

Figure 3-7

Figure (a) shows a typical use of PCI in a single-processor system. Figure (b) shows a typical use of PCI in a multiprocessor system. Notice that more than one PCI configuration may be used.

Check your progress

Q1. Explain

(i) Data lines
(ii) Address lines
(iii)Control lines

Q2. Define elements of Bus Design

Q3. Explain Data Transfer Type.

3.2 Processor to Memory Communication

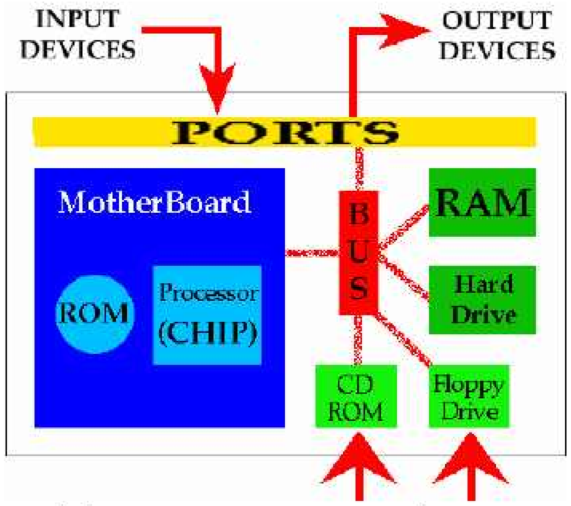


Figure 3-8 shows the parts of a computer:

- The Central Processing Unit:
  - (CPU),
  - Buses,
  - Ports and controllers,
  - ROM;
- Main Memory (RAM);
- Input Devices;
- Output Devices;
- Secondary Storage;
  - floppy disks,
  - hard disk,
  - CD-ROM

This part of the reading will examine the CPU, Buses, Controllers, and Main Memory. Other sections will examine input devices, output devices, and secondary memory.

The Central Processing Unit (CPU)

The computer does its primary work in a part of the machine we cannot see, a control center that converts data input to information output. This control center, called the central processing unit (CPU), is a highly complex, extensive set of electronic circuitry that executes stored program instructions. All computers, large and small, must have a central processing unit. As Figure 1 shows, the central processing unit consists of two parts: The control unit and the arithmetic/logic unit. Each part has a specific function.

Before we discuss the control unit and the arithmetic/logic unit in detail, we need to consider data storage and its relationship to the central processing unit. Computers use two types of storage: Primary storage and secondary storage. The CPU interacts closely with primary storage, or main memory, referring to it for both instructions and data. For this reason this part of the reading will discuss memory in the context of the central processing unit. Technically, however, memory is not part of the CPU.

Recall that a computer's memory holds data only temporarily, at the time the computer is executing a program. Secondary storage holds permanent or semi-permanent data on some external magnetic or optical medium. The diskettes and CD-ROM disks that you have seen with personal computers are secondary storage devices, as are hard disks. Since the physical attributes of secondary storage devices determine the way data is organized on them, we will discuss secondary storage and data organization together in another part of our on-line readings.

Now let us consider the components of the central processing unit.

The Control Unit

The control unit of the CPU contains circuitry that uses electrical signals to direct the entire computer system to carry out, or execute, stored program instructions. Like an orchestra leader, the control unit does not execute program instructions; rather, it directs other parts of the system to do so. The control unit must communicate with both the arithmetic/logic unit and memory.

The Arithmetic/Logic Unit

The arithmetic/logic unit (ALU) contains the electronic circuitry that executes all arithmetic and logical operations.

The arithmetic/logic unit can perform four kinds of arithmetic operations, or mathematical calculations: addition, subtraction, multiplication, and division. As its name implies, the arithmetic/logic unit also performs logical operations. A logical operation is usually a comparison. The unit can compare numbers, letters, or special characters. The computer can then take action based on the result of the comparison. This is a very important capability. It is by comparing that a computer is able to tell, for instance, whether there are unfilled seats on airplanes, whether charge- card customers have exceeded their credit limits, and whether one candidate of a party has more votes than another.

Logical operations can test for three conditions:

- **Equal-to condition.** In a test for this condition, the arithmetic/logic unit compares two values to determine if they are equal. For example: If the number of tickets sold equals the number of seats in the auditorium, then the concert is declared sold out.
- **Less-than condition.** To test for this condition, the computer compares values to determine if one is less than another. For example: If the number of speeding tickets on a driver's record is less than three, then insurance rates are $425; otherwise, the rates are $500.
- **Greater-than condition.** In this type of comparison, the computer determines if one value is greater than another. For example: If the hours a person worked this week are greater than 40, then multiply every extra hour by 1.5 times the usual hourly wage to compute overtime pay.

A computer can simultaneously test for more than one condition. In fact, a logic unit can usually discern six logical relationships: *equal to, less than, greater than, less than or equal to, greater than or equal to,* and *not equal.*

The symbols that let you define the type of comparison you want the computer to perform are called relational operators. The most common relational operators are the equal sign(=), the less-than symbol(<), and the greater-than symbol(>).

- **Registers: Temporary Storage Areas**

  Registers are temporary storage areas for instructions or data. They are not a part of memory; rather they are special additional storage locations that offer the advantage of speed. Registers work under the direction of the control unit to accept, hold, and transfer instructions or data and perform arithmetic or logical comparisons at high speed. The control unit uses a data storage register the way a store owner uses a cash register-as a temporary, convenient place to store what is used in transactions.

  Computers usually assign special roles to certain registers, including these registers:

  - **An accumulator**, which collects the result of computations.
  - **An address register**, which keeps track of where a given instruction or piece of data is stored in memory. Each storage location in memory is identified by an address, just as each house on a street has an address.
  - **A storage register**, which temporarily holds data taken from or about to be sent to memory.
  - A general-purpose **register**, which is used for several functions.

- **Memory and Storage**

Memory is also known as primary storage, primary memory, main storage, internal storage, main memory, and RAM (Random Access Memory); all these terms are used interchangeably by people in computer circles. Memory is the part of the computer that holds data and instructions for processing. Although closely associated with the central processing unit, memory is separate from it. Memory stores program instructions or data for only as long as the program they pertain to is in operation. Keeping these items in memory when the program is not running is not feasible for three reasons:

- Most types of memory only store items while the computer is turned on; data is destroyed when the machine is turned off.

- If more than one program is running at once (often the case on large computers and sometimes on small computers), a single program can not lay exclusive claim to memory.
- There may not be room in memory to hold the processed data.

How do data and instructions get from an input device into memory? The control unit sends them. Likewise, when the time is right, the control unit sends these items from memory to the arithmetic/logic unit, where an arithmetic operation or logical operation is performed. After being processed, the information is sent to memory, where it is hold until it is ready to be released to an output unit.

The chief characteristic of memory is that it allows very fast access to instructions and data, no matter where the items are within it.

To see how registers, memory, and second storage all work together, let us use the analogy of making a salad. In our kitchen we have:

- a refrigerator where we store our vegetables for the salad;
- a counter where we place all of our veggies before putting them on the cutting board for chopping;
- a cutting board on the counter where we chop the vegetables;
- a recipe that details what veggies to chop;
- the corners of the cutting board are kept free for partially chopped piles of veggies that we intend to chop more or to mix with other partially chopped veggies.
- a bowl on the counter where we mix and store the salad;
- space in the refrigerator to put the mixed salad after it is made.

The process of making the salad is then: bring the veggies from the fridge to the counter top; place some veggies on the chopping board according to the recipe; chop the veggies, possibly storing some partially chopped veggies temporarily on the corners of the cutting board; place all the veggies in the bowl to either put back in the fridge or put directly on the dinner table.

The refrigerator is the equivalent of secondary (disk) storage. It can store high volumes of veggies for long periods of time. The counter top is the equivalent of the computer's motherboard - everything is done on the counter (inside the computer). The cutting board is the ALU - the work gets done there. The recipe is the control unit - it tells you what to do on the cutting board (ALU). Space on the counter top is the equivalent of RAM memory - all veggies must be brought from the fridge and placed on the counter top for fast access. Note that the counter top (RAM) is faster to access than the fridge (disk), but cannot hold as much, and cannot hold it for long periods of time. The corners of the cutting board where we temporarily store partially chopped veggies are equivalent to the registers. The corners of the cutting board are very fast to access for chopping, but cannot hold much. The salad bowl is like a temporary register, it is for storing the salad waiting to take back to the fridge (putting data back on a disk) or for taking to the dinner table (outputting the data to an output device).

Now for a more technical example, let us look at how a payroll program uses all three types of storage. Suppose the program calculates the salary of an employee. The data representing the hours worked and the data for the rate of pay are ready in their respective registers. Other data related to the salary calculation-overtime hours, bonuses, deductions, and so forth-is waiting nearby in memory. The data for other employees is available in secondary storage. As the CPU finishes calculations about one employee, the data about the next employee is brought from secondary storage into memory and eventually into the registers.

The following table summarizes the characteristics of the various kinds of data storage in the storage hierarchy.

Storage	**Speed**	**Capacity**	**Relative Cost ($)**	**Permanent?**
Registers	Fastest	Lowest	Highest	No
RAM	Very Fast	Low/Moderate	High	No
Floppy Disk	Very Slow	Low	Low	Yes
Hard Disk	Moderate	Very High	Very Low	Yes

Modern computers are designed with this hierarchy due to the characteristics listed in the table. It has been the cheapest way to get the functionality. However, as RAM becomes cheaper, faster, and even permanent, we may see disks disappear as an internal storage device. Removable disks, like Zip disks or CDs (we describe these in detail in the online reading on storage devices) will probably remain in use longer as a means to physically transfer large volumes of data into the computer. However, even this use of disks will probably be supplanted by the Internet as the major (and eventually only) way of transferring data. Floppy disks drives are already disappearing: the new IMac Macintosh from Apple does not come with one. Within the next five years most new computer designs will only include floppy drives as an extra for people with old floppy disks that they must use.

3.3 I/O to Processor Communication

Input/output

Buses: Connecting I/O to Processor and Memory

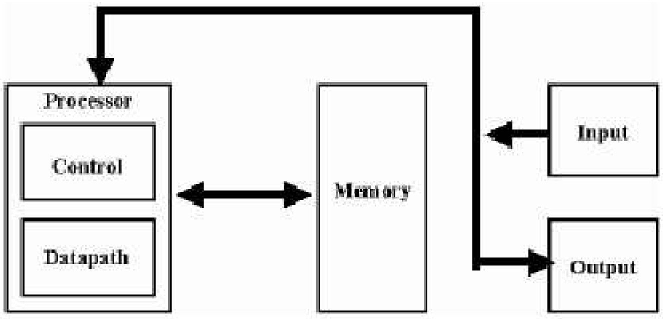


Figure 3-9

- A bus is a shared communication link
- It uses one set of wires to connect multiple subsystems

Sometimes shared bus with memory, sometimes a separate I/O bus

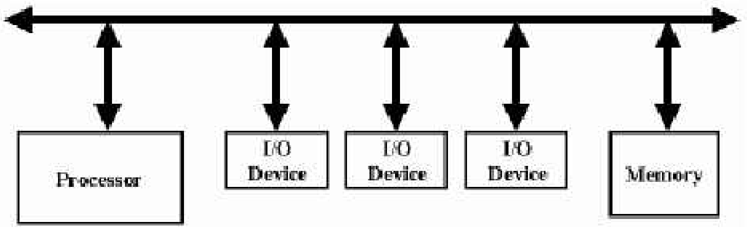


Figure 3-10

Advantages

- Versatility:
- New devices can be added easily
- Peripherals can be moved between computer systems that use the same bus standard
- Low Cost:
- A single set of wires is shared in multiple ways

Disadvantages

- It creates a communication bottleneck
- The bandwidth of that bus can limit the maximum I/O throughput
- The maximum bus speed is largely limited by:
- The length of the bus
- The number of devices on the bus
- The need to support a range of devices with:
    - Widely varying latencies
    - Widely varying data transfer rates

Synchronous and Asynchronous Bus

- Synchronous Bus:
- Includes a clock in the control lines
- A fixed protocol for communication that is relative to the clock
- Advantage: involves very little logic and can run very fast
- Disadvantages:
    - Every device on the bus must run at the same clock rate
    - To avoid clock skew, they cannot be long if they are fast
- Asynchronous Bus:

- It is not clocked
- It can accommodate a wide range of devices
- It can be lengthened without worrying about clock skew
- It requires a handshaking protocol

A Handshaking Protocol

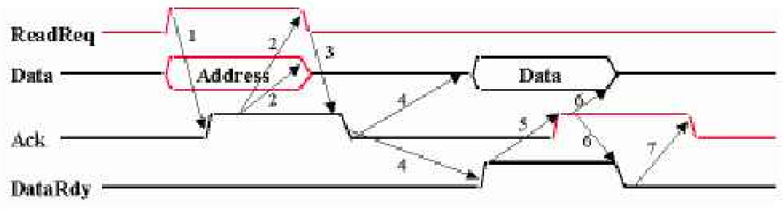


Figure 3-11

- Three control lines
- ReadReq: indicate a read request for memory

  Address is put on the data lines at the same time
- DataReady: indicate the data word is now ready on the data lines

  Data is put on the data lines at the same time
- Ack: acknowledge the ReadReq or the DataRdy of the other party

Increasing the Bus Bandwidth

- Data bus width:
- By increasing the width of the data bus, transfers of multiple words require fewer bus cycles
- Example: SPARCstation 20?s memory bus is 128 bit wide
- Cost: more bus lines
- Block transfers:
- Allow the bus to transfer multiple words in back-to-back bus cycles
- Only one address needs to be sent at the beginning
- The bus is not released until the last word is transferred
- Cost: (a) increased complexity (b) decreased response time for request

Bus Arbitration

Any device that can control the bus is called a bus master

- Bus arbitration scheme:
- A bus master wanting to use the bus asserts the bus request
- A bus master cannot use the bus until its request is granted
- A bus master must signal to the arbiter after it has finished using the bus
- Bus arbitration schemes usually try to balance two factors:
- Bus priority: the highest priority device should be serviced first
- Fairness: Even the lowest priority device should never be completely locked out from the bus

Giving Commands to I/O Devices

- Two methods are used to address the device:
- Special I/O instructions
- Memory-mapped I/O
- Special I/O instructions specify:
- Both the device number and the command word
  - Device number: the processor communicates this via a set of wires normally included as part of the I/O bus
  - Command word: this is usually send on the bus
- Memory-mapped I/O:
- Portions of the address space are assigned to I/O device
- Read and writes to those addresses are interpreted as commands to the I/O devices

Notifying the OS

- The OS needs to know when:
- The I/O device has completed an operation
- The I/O operation has encountered an error
- This can be accomplished in two different ways:

- Poling:
    - The I/O device put information in a status register
    - The OS periodically check the status register
- I/O Interrupt
    - Whenever an I/O device needs attention from the processor, it interrupts the processor.

Polling

Check the device a regular intervals.

Advantage:

- Simple: the processor is totally in control and does all the work
- Disadvantage:
- Polling overhead can consume a lot of CPU time

Interrupt Driven Data Transfer

- Advantage:
- User program progress is only halted during actual transfer
- Disadvantage, special hardware is needed to:
- Cause an interrupt (I/O device)
- Detect an interrupt (processor)
- Save the proper states to resume after the interrupt (processor)

Delegating I/O Responsibility from the CPU: DMA

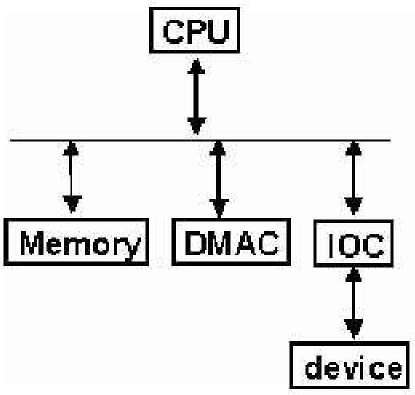


Figure 3-12

- Direct Memory Access (DMA):
- External to the CPU
- Act as a maser on the bus
- Transfer blocks of data to or from memory without CPU intervention

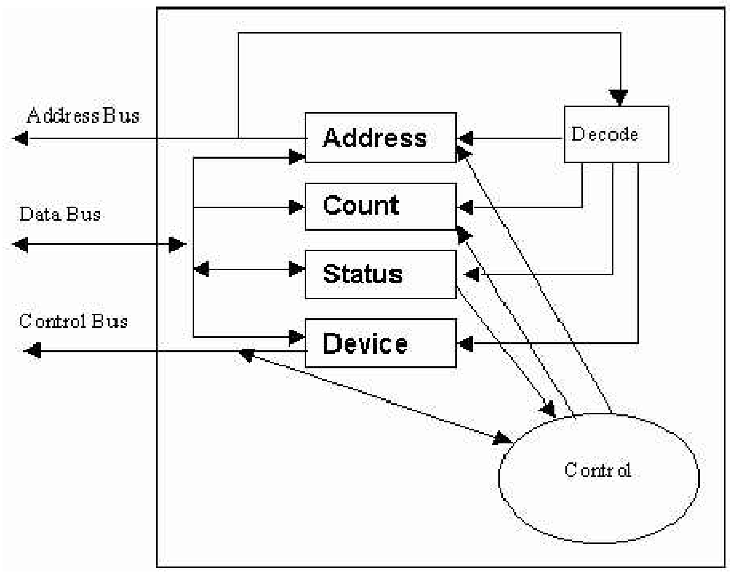


Figure 3-13 Block Diagram of a DMA Controller

Sequence of operations - Input

1. CPU loads registers - Address, Count and Device
2. CPU Sets Status to Input
3. Controller asks for bus
4. Gets bus - does read from device, puts Address on the bus, does write to memory and decrements count.
   Repeat 4 until count=0
5. Interrupt CPU to signal end of transfer

Output is similar but status is set to Output and data is read from memory and written to the output device.

Check your progress

Q1. Explain three conditions of Logical operations

Q2. How do data and instructions get from an input device into memory?

Q3. Define Polling.

3.4 Summary

In this unit you learnt about Interconnection of Units, Processor to Memory communication, I/O to Processor Communication. You also learn the working of buses and how they established communication with one another.

- A computer consists of a set of components or modules (processor, memory, I/O) that communicate with each other.
- A bus is a communication pathway connecting two or more devices.
- Multiple devices can be connected to the same bus at the same time.
- A bus consists of 50 to hundreds of separate lines.
- Memory is also known as primary storage, primary memory, main storage, internal storage, main memory, and RAM (Random Access Memory);

3.5 Review Questions

Q1. What do you mean by interconnection of units? Explain with example.

Q2. What do you mean by a bus? Explain in detail.

Q3. Define the working of a bus?

Q4. How processors establish communication to memory? Elaborate your answer.

Q5. How processors establish communication to I/O Units? Elaborate your answer.

UNIT-IV

Multiprogramming

Structure

4.0 Introduction

4.1 Interrupt Structures

4.2 Multiprogramming

4.3 Processor Features

4.4 Reduced Instruction Set Computers (RISC)

4.5 Virtual memory

4.6 Summary

4.7 Review Questions

4.0 Introduction

In this unit we focused on multiprogramming. There are seven sections in this unit. In the first sections i.e. 4.1 you will know about interrupt structures. Interrupt is signals send by an external device to the processor, to request the processor to perform a particular task or work. Mainly in the microprocessor based system the interrupts are used for data transfer between the peripheral and the microprocessor. The processor will check the interrupts always at the 2nd T-state of last machine cycle. If there is any interrupt it accept the interrupt and send the INTA (active low) signal to the peripheral. The vectored address of particular interrupt is stored in program counter. The processor executes an interrupt service routine (ISR) addressed in program counter. It returned to main program by RET instruction. In the Sec. 4.2 you will learn about multiprogramming. Multiprogramming is a rudimentary form of parallel processing in which several programs are run at the same time on a uniprocessor. Since there is only one processor, there can be no true simultaneous execution of different programs. In Sec. 4.3 we defined processor features. In Sec. 4.4 we explained Reduced Instruction Set Computers (RISC). Reduced instruction set computing, or RISC (pronounced 'risk'), is a CPU design strategy based on the insight that a simplified instruction set provides higher performance when combined with a microprocessor architecture capable of executing those instructions using fewer microprocessor cycles per instruction. In sec. 4.5 you will know about virtual memory. In Sec. 4.6 and 4.7 you will find summary and review questions respectively.

Objectives

After studying this unit you should be able to:

- Define Interrupt Structures, Multiprogramming
- Describe Processor Features, Reduced Instruction Set Computers (RISC), and Virtual memory.

4.1 Interrupt Structures

Interrupt structure refers to the precedence of interrupts. Hardware events that cause interrupts are assigned CPU interrupt levels. The CPU can disable interrupts of a certain level and below, thus allowing an important interrupt to preempt an interrupt of lower priority, but not vice-versa. Most machine architectures do not allow the software to reconfigure the interrupt structure. If the precedence of interrupts can be reconfigured, though, make sure that the clock that is used for task-switching has its interrupts serviced at a high priority so that lower priority events (such as disk access) are not capable of disabling the pre-emptive scheduling. The serial port should also have a high priority so that high baud rates can be reliably used.

The precedence of interrupts is important because disabling interrupts is a common way to provide mutual exclusion in the kernel. For example, a routine that modifies tty structures (those that handle input and output to a terminal) will call spltty() (for Set Processor Level TTY) just before a critical section and later will reset the level to what it was before spltty() was called. On most machines, this call will disable any interrupt that would call a function that handles the tty structures and, in addition, will disable any interrupts with a priority lower than that of "tty". An interrupt structure not well suited for UNIX might cause relatively long interrupt code (*e.g.*, disk access or sound driver processing) to block more frequent interrupts (*e.g.*, timer, keyboard, or serial port). On the Macintosh port, heavy use of the hard disk would cause the system time to be off by several hours a day, and scrolling of the display, being in a tty device, would preempt the keyboard routine and cause keystrokes to be lost. On A/UX, Apple's System V UNIX for the Macintosh, a beep causes serial port data to be lost.

4.2 Multiprogramming

Early computers ran one process at a time. While the process waited for servicing by another device, the CPU was idle. In an I/O intensive process, the CPU could be idle as much as 80% of the time. Advancements in operating systems led to computers that load several independent processes into memory and switch the CPU from one job to another when the first becomes blocked while waiting for servicing by another device. This idea of *multiprogramming* reduces the idle time of the CPU. Multiprogramming accelerates the throughput of the system by efficiently using the CPU time.

Programs in a multiprogrammed environment appear to run at the same time. Processes running in a multiprogrammed environment are called *concurrent processes.* In actuality, the CPU processes one instruction at a time, but can execute instructions from any active process.

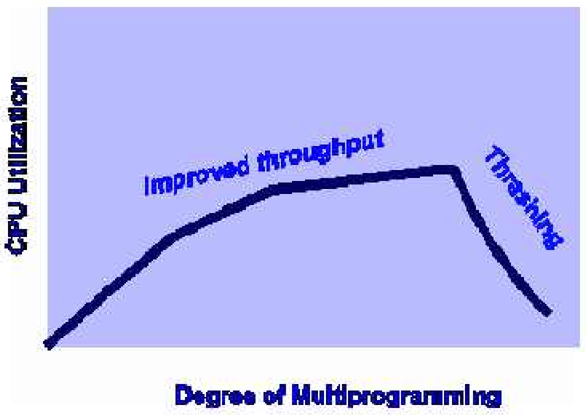


Figure 4-1

As the illustration shows, CPU utilization of a system can be improved by using multiprogramming. Let P be the fraction of time that a process spends away from the CPU. If there is one process in memory, the CPU utilization is (1-P). If there are N processes in memory, the probability of N processes waiting for an I/O is P*P...*P (N times). The CPU utilization is (1-P^N) where N is called the *multiprogramming level* (MPL) or the *degree of multiprogramming.* As N increases, the CPU utilization increases. While this equation indicates that a CPU continues to work more efficiently as more and more processes are added, logically, this cannot be true. Once the system passes the point of optimal CPU utilization, it thrashes.

In order to use the multiprogramming concept, processes must be loaded into independent sections or *partitions* of memory. So, main memory is divided into fixed-sized or variable-sized partitions. Since a partition may not be large enough for the entire process, virtual memory is implemented to keep the processes executing. The answers to several questions are important to implementing an efficient virtual memory system in a multiprogrammed environment.

4.3 Processor Features

As new processors are introduced, new features are continually added to their architectures to help improve everything from performance in specific types of applications to the reliability of the CPU as a whole. The next few sections take a look at some of these technologies, including System Management Mode (SMM), Superscalar Execution, MMX, SSE, 3DNow, HT Technology, and dual-core processing.

SMM (Power Management)

Spurred on primarily by the goal of putting faster and more powerful processors in laptop computers, Intel has created power-management circuitry. This circuitry enables processors to conserve energy use and lengthen battery life. This was introduced initially in the Intel 486SL processor, which is an enhanced version of the 486DX processor. Subsequently, the power-management features were universalized and incorporated into all 75MHz and faster Pentium and later processors. This feature set is called SMM, which stands for *system management mode*.

SMM circuitry is integrated into the physical chip but operates independently to control the processor's power use based on its activity level. It enables the user to specify time intervals after which the CPU will be partially or fully powered down. It also supports the Suspend/Resume feature that allows for instant power on and power off, used mostly with laptop PCs. These settings are typically controlled via system BIOS settings.

Superscalar Execution

The fifth-generation Pentium and newer processors feature multiple internal instruction execution pipelines, which enable them to execute multiple instructions at the same time. The 486 and all preceding chips can perform only a single instruction at a time. Intel calls the capability to execute more than one instruction at a time *superscalar* technology. This technology provides additional performance compared with the 486.

Superscalar architecture usually is associated with high-output Reduced Instruction Set Computer (RISC) chips. A RISC chip has a less complicated instruction set with fewer and simpler instructions. Although each instruction accomplishes less, overall the clock speed can be higher, which can usually increase performance. The Pentium is one of the first Complex Instruction Set Computer (CISC) chips to be considered superscalar. A CISC chip uses a richer, fuller-featured instruction set, which has more complicated instructions. As an example, say you wanted to instruct a robot to screw in a light bulb. Using CISC instructions, you would say

1. Pick up the bulb.
2. Insert it into the socket.
3. Rotate clockwise until tight.

Using RISC instructions, you would say something more along the lines of

1. Lower hand.
2. Grasp bulb.
3. Raise hand.
4. Insert bulb into socket.
5. Rotate clockwise one turn.
6. Is bulb tight? If not, repeat step 5.
7. End.

Overall, many more RISC instructions are required to do the job because each instruction is simpler (reduced) and does less. The advantage is that there are fewer overall commands the robot (or processor) has to deal with and it can execute the individual commands more quickly, and thus in many cases execute the complete task (or program) more quickly as well. The debate goes on whether RISC or CISC is really better, but in reality there is no such thing as a pure RISC or CISC chip–it is all just a matter of definition, and the lines are somewhat arbitrary.

Intel and compatible processors have generally been regarded as CISC chips, although the fifth-and sixth-generation versions have many RISC attributes and internally break CISC instructions down into RISC versions.

MMX Technology

MMX technology was originally named for multimedia extensions, or matrix math extensions, depending on whom you ask. Intel officially states that it is actually not an abbreviation and stands for nothing other than the letters MMX (not being an abbreviation was apparently required so that the letters could be trademarked); however, the internal origins are probably one of the preceding. MMX technology was introduced in the later fifth-generation Pentium processors as a kind of add-on that improves video compression/decompression, image manipulation, encryption, and I/O processing–all of which are used in a variety of today's software.

MMX consists of two main processor architectural improvements. The first is very basic; all MMX chips have a larger internal L1 cache than their non-MMX counterparts. This improves the performance of any and all software running on the chip, regardless of whether it actually uses the MMX-specific instructions.

The other part of MMX is that it extends the processor instruction set with 57 new commands or instructions, as well as a new instruction capability called single instruction, multiple data (SIMD).

Modern multimedia and communication applications often use repetitive loops that, while occupying 10% or less of the overall application code, can account for up to 90% of the execution time. SIMD enables one instruction to perform the same function on multiple pieces of data, similar to a teacher telling an entire class to "sit down," rather than addressing each student one at a time. SIMD enables the chip to reduce processor-intensive loops common with video, audio, graphics, and animation.

Intel also added 57 new instructions specifically designed to manipulate and process video, audio, and graphical data more efficiently. These instructions are oriented to the *highly parallel* and often repetitive sequences frequently found in multimedia operations. *Highly parallel* refers to the fact that the same processing is done on many data points, such as when modifying a graphic image. The main drawbacks to MMX were that it worked only on integer values and used the floating-point unit for processing, so time was lost when a shift to floating-point operations was necessary. These drawbacks were corrected in the additions to MMX from Intel and AMD.

Intel licensed the MMX capabilities to competitors such as AMD and Cyrix, who were then able to upgrade their own Intel-compatible processors with MMX technology.

SSE, SSE2, and SSE3

In February 1999, Intel introduced the Pentium III processor and included in that processor an update to MMX called Streaming SIMD Extensions (SSE). These were also called Katmai New Instructions (KNI) up until their debut because they were originally included on the Katmai processor, which was the codename for the Pentium III. The Celeron 533A and faster Celeron processors based on the Pentium III core also support SSE instructions. The earlier Pentium II and Celeron 533 and lower (based on the Pentium II core) do not support SSE.

SSE includes 70 new instructions for graphics and sound processing over what MMX provided. SSE is similar to MMX; in fact, besides being called KNI, SSE was also called MMX-2 by some before it was released. In addition to adding more MMX style instructions, the SSE instructions allow for floating-point calculations and now use a separate unit within the processor instead of sharing the standard floating-point unit as MMX did.

SSE2 was introduced in November 2000, along with the Pentium 4 processor, and adds 144 additional SIMD instructions. SSE2 also includes all the previous MMX and SSE instructions.

SSE3 was introduced in February 2004, along with the Pentium 4 Prescott processor, and adds 13 new SIMD instructions to improve complex math, graphics, video encoding, and thread synchronization. SSE3 also includes all the previous MMX, SSE, and SSE2 instructions.

The Streaming SIMD Extensions consist of new instructions, including SIMD floating-point, additional SIMD integer, and cache ability control instructions. Some of the technologies that benefit from the Streaming SIMD Extensions include advanced imaging, 3D video, streaming audio and video (DVD playback), and speech-recognition applications. The benefits of SSE include the following:

- Higher resolution and higher quality image viewing and manipulation for graphics software
- High-quality audio, MPEG2 video, and simultaneous MPEG2 encoding and decoding for multimedia applications
- Reduced CPU utilization for speech recognition, as well as higher accuracy and faster response times when running speech-recognition software

The SSEx instructions are particularly useful with MPEG2 decoding, which is the standard scheme used on DVD video discs. SSE-equipped processors should therefore be more capable of performing MPEG2 decoding in software at full speed without requiring an additional hardware MPEG2 decoder card. SSE-equipped processors are much better and faster than previous processors when it comes to speech recognition, as well.

One of the main benefits of SSE over plain MMX is that it supports single-precision floating-point SIMD operations, which have posed a bottleneck in the 3D graphics processing. Just as with plain MMX, SIMD enables multiple operations to be performed per processor instruction. Specifically, SSE supports up to four floating-point operations per cycle; that is, a single instruction can operate on four pieces of data simultaneously. SSE floating-point instructions can be mixed with MMX instructions with no performance penalties. SSE also supports data *prefetching*, which is a mechanism for reading data into the cache before it is actually called for.

Note that for any of the SSE instructions to be beneficial, they must be encoded in the software you are using, so SSE-aware applications must be used to see the benefits. Most software companies writing graphics-and sound-related software today have updated those applications to be SSE aware and use the features of

SSE. For example, high-powered graphics applications such as Adobe Photoshop support SSE instructions for higher performance on processors equipped with SSE. Microsoft includes support for SSE in its DirectX 6.1 and later video and sound drivers, which are included with Windows 98 Second Edition, Windows Me, and Windows NT 4.0 (with service pack 5 or later), Windows 2000, and Windows XP.

SSE is an extension to MMX; SSE2 is an extension to SSE; and SSE3 is an extension to SSE2. Therefore, processors that support SSE3 also support the SSE2 instructions, processors that support SSE2 also support SSE, and processors that support SSE also support the original MMX instructions. This means that standard MMX-enabled applications run as they did on MMX-only processors.

The first AMD processors to support SSE3 are the 0.09-micron versions of the Athlon 64 and all versions of the dual-core Athlon 64 X2.

3DNow Enhanced 3DNowand Professional 3DNow

3DNow Technology was originally introduced as AMD's alternative to the SSE instructions in the Intel processors. Actually, 3DNow was first introduced in the K6 series before Intel released SSE in the Pentium III, and then AMD added Enhanced 3DNow to the Athlon and Duron processors. The latest version, Professional 3DNow, was introduced in the first Athlon XP processors. AMD licensed MMX from Intel, and all its K6 series, Athlon, Duron, and later processors include full MMX instruction support. Not wanting to additionally license the SSE instructions being developed by Intel, AMD first came up with a different set of extensions beyond MMX called 3DNow. Introduced in May 1998 in the K6-2 processor and enhanced when the Athlon was introduced in June 1999, 3DNow, and Enhanced 3DNow are sets of instructions that extend the multimedia capabilities of the AMD chips beyond MMX. This enables greater performance for 3D graphics, multimedia, and other floating-point-intensive PC applications.

3DNow Technology is a set of 21 instructions that uses SIMD techniques to operate on arrays of data rather than single elements. Enhanced 3DNow adds 24 more instructions (19 SSE and 5 DSP/communications instructions) to the original 21 for a total of 45 new instructions. Positioned as an extension to MMX technology, 3DNow is similar to the SSE found in the Pentium III and Celeron processors from Intel. According to AMD, 3DNow provides approximately the same level of improvement to MMX as did SSE, but in fewer instructions with less complexity. Although similar in capability, they are not compatible at the instruction level, so software specifically written to support SSE does not support 3DNow, and vice versa. The latest version of 3DNow, 3DNow Professional, adds 51 SSE commands to 3DNow Enhanced, meaning that 3DNow Professional now supports all SSE commands, meaning that AMD chips now essentially have SSE capability. Unfortunately, AMD includes SSE2 only on the Athlon 64, Athlon 64FX, and Opteron 64-bit processors.

Just as with SSE, 3DNowalso supports single precision floating-point SIMD operations and enables up to four floating-point operations per cycle. 3DNow floating-point instructions can be mixed with MMX instructions with no performance penalties. 3DNow also supports data prefetching.

Also like SSE, 3DNow is well supported by software, including Windows 9x, Windows NT 4.0, and all newer Microsoft operating systems. 3DNow-specific support is no longer a big issue if you are using an Athlon XP or Athlon 64 processor because they now fully support SSE through their support of 3DNow Professional.

Dynamic Execution

First used in the P6 or sixth-generation processors, dynamic execution enables the processor to execute more instructions on parallel, so tasks are completed more quickly. This technology innovation is comprised of three main elements:

- *Multiple branch prediction.* Predicts the flow of the program through several branches
- *Dataflow analysis.* Schedules instructions to be executed when ready, independent of their order in the original program
- *Speculative execution.* Increases the rate of execution by looking ahead of the program counter and executing instructions that are likely to be necessary

Branch Prediction

Branch prediction is a feature formerly found only in high-end mainframe processors. It enables the processor to keep the instruction pipeline full while running at a high rate of speed. A special fetch/decode unit in the processor uses a highly optimized branch prediction algorithm to predict the direction and outcome of the instructions being executed through multiple levels of branches, calls, and returns. It is similar to a chess player working out multiple strategies in advance of game play by predicting the opponent's strategy several moves into the future. By predicting the instruction outcome in advance, the instructions can be executed with no waiting.

Dataflow Analysis

Dataflow analysis studies the flow of data through the processor to detect any opportunities for out-of-order instruction execution. A special dispatch/execute unit in the processor monitors many instructions and can execute these instructions in an order that optimizes the use of the multiple superscalar execution units. The resulting out-of-order execution of instructions can keep the execution units busy even when cache misses and other data-dependent instructions might otherwise hold things up.

Speculative Execution

Speculative execution is the processor's capability to execute instructions in advance of the actual program counter. The processors dispatch/execute unit uses dataflow analysis to execute all available instructions in the instruction pool and store the results in temporary registers. A retirement unit then searches the instruction pool for completed instructions that are no longer data dependent on other instructions to run or which have unresolved branch predictions. If any such completed instructions are found, the results are committed to memory by the retirement unit or the appropriate standard Intel architecture in the order they were originally issued. They are then retired from the pool.

Dynamic execution essentially removes the constraint and dependency on linear instruction sequencing. By promoting out-of-order instruction execution, it can keep the instruction units working rather than waiting for data from memory. Even though instructions can be predicted and executed out of order, the results are committed in the original order so as not to disrupt or change program flow. This enables the P6 to run existing Intel architecture software exactly as the P5 (Pentium) and previous processors did–just a whole lot more quickly!

Dual Independent Bus Architecture

The Dual Independent Bus (DIB) architecture was first implemented in the sixth-generation processors from Intel and AMD. DIB was created to improve processor bus bandwidth and performance. Having two (dual) independent data I/O buses enables the processor to access data from either of its buses simultaneously and in parallel, rather than in a singular sequential manner (as in a single-bus system). The main (often called front-side) processor bus is the interface between the processor and the motherboard or chipset. The second (back-side) bus in a processor with DIB is used for the L2 cache, enabling it to run at much greater speeds than if it were to share the main processor bus.

Two buses make up the DIB architecture: the L2 cache bus and the main CPU bus, often called FSB (front-side bus). The P6 class processors from the Pentium Pro to the Celeron, Pentium II/III/4, and Athlon/Duron processors can use both buses simultaneously, eliminating a bottleneck there. The dual bus architecture enables the L2 cache of the newer processors to run at full speed inside the processor core on an independent bus, leaving the main CPU bus (FSB) to handle normal data flowing in and out of the chip. The two buses run at different speeds. The front-side bus or main CPU bus is coupled to the speed of the motherboard, whereas the back-side or L2 cache bus is coupled to the speed of the processor core. As the frequency of processors increases, so does the speed of the L2 cache.

The key to implementing DIB was to move the L2 cache memory off the motherboard and into the processor package. L1 cache always has been a direct part of the processor die, but L2 was larger and originally had to be external. By moving the L2 cache into the processor, the L2 cache could run at speeds more like the L1 cache, much faster than the motherboard or processor bus.

DIB also enables the system bus to perform multiple simultaneous transactions (instead of singular sequential transactions), accelerating the flow of information within the system and boosting performance. Overall, DIB architecture offers up to three times the bandwidth performance over a single-bus architecture processor.

Hyper-Threading Technology

Computers with two or more physical processors have long had a performance advantage over single-processor computers when the operating system supported multiple processors, as is the case with Windows NT 4.0, 2000, XP Professional, and Linux. However, dual-processor motherboards and systems have always been more expensive than otherwise-comparable single processor systems, and upgrading a dual-processor-capable system to dual-processor status can be difficult with only one processor because of the need to match processor speeds and specifications. However, Intel's Hyper-Threading (HT) Technology allows a single processor to handle two independent sets of instructions at the same time. In essence, HT Technology converts a single physical processor into two virtual processors.

Intel originally introduced HT Technology in its line of Xeon processors for servers in March 2002. HT Technology enables multiprocessor servers to act as if they had twice as many processors installed. HT Technology was introduced on Xeon workstation-class processors with a 533MHz system bus and later found its way into PC processors, with the Pentium 4 3.06GHz processor in November 2002. HT Technology is also present in all Pentium 4 processors with 800MHz CPU bus speed (2.4GHz up through 3.8GHz) as well as the Pentium 4 Extreme Edition and the dual-core Pentium Extreme Edition. However, the dual-core Pentium D does not include HT Technology.

How Hyper-Threading Works

Internally, an HT-enabled processor has two sets of general-purpose registers, control registers, and other architecture components, but both logical processors share the same cache, execution units, and buses.

Although the sharing of some processor components means that the overall speed of an HT-enabled system isn't as high as a true dual-processor system would be, speed increases of 25% or more are possible when multiple applications or a single multithreaded application is being run.

Hyper-Threading Requirements

The first HT-enabled processor was the Intel Pentium 4 3.06GHz. All 3.06GHz and faster Pentium 4 models support HT Technology, as do all processors 2.4GHz and faster that use the 800MHz bus. However, an HT-enabled P4 processor by itself can't bring the benefits of HT Technology to your system. You also need the following:

- *A compatible motherboard (chipset). It might need a BIOS upgrade.*
- *BIOS support to enable/disable HT Technology. If your operating system doesn't support HT Technology, you should disable this feature. Application performance varies (some faster, some slower) when HT Technology is enabled. If this is a matter of concern, you should perform*

application-based benchmarks with HT Technology enabled and disabled to determine whether your application mix will benefit from using HT Technology.

- *A compatible operating system such as Windows XP.* When hyper-threading is enabled, the Device Manager shows two processors.

Intel's newer chipsets for the Pentium 4 support HT Technology. However, if your motherboard or computer was released before HT Technology was introduced, you will need a BIOS upgrade from the motherboard or system vendor to be able to use HT Technology. Although Windows NT 4.0 and Windows 2000 are designed to use multiple physical processors, HT Technology requires specific operating system optimizations to work correctly. Linux distributions based on kernel 2.4.18 and higher also support HT Technology.

Dual-core Technology

HT Technology is designed to simulate two processors in a single physical unit. With properly written software, HT Technology can improve application performance. Unfortunately, many applications do not support HT Technology and slow down when HT Technology is enabled. However, applications do not need to be rewritten to take advantage of multiple processors or dual-core processors. A dual-core processor, as the name implies, contains two processor cores in a single processor package. A dual-core processor provides virtually all the advantages of a multiple-processor computer at a cost lower than two matched processors.

Both AMD and Intel introduced dual-core x86-compatible desktop processors in 2005. AMD's entry–the Athlon 64 X2–can be installed in most Socket 939 motherboards designed for the original single-core Athlon 64 or Athlon 64 FX processors. A BIOS upgrade might be necessary in some situations. AMD also introduced dual-core versions of the Opteron workstation and server processor in 2005. Intel's first dual-core processors–the Pentium Extreme Edition and the Pentium D–use the same Socket 775 as the most recent Pentium 4 models. However, they require new motherboards using the Intel 945 and 955 series chipsets or third-party chipsets that support dual-core operation.

Check your progress

Q1. Define Interrupt Structures

Q2. Explain degree of multiprogramming

Q3. How Hyper-Threading Works?

4.3 Reduced instruction set computing (RISC)

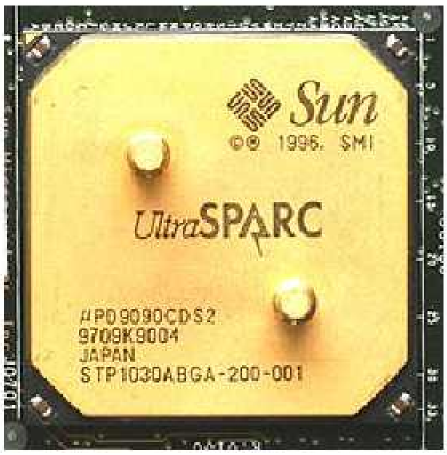


Figure 4-2

Reduced Instruction set computing, is a CPU design strategy based on the insight that simplified (as opposed to complex) instructions can provide higher performance if this simplicity enables much faster execution of each instruction. A computer based on this strategy is a *reduced instruction set computer*, also called *RISC*. The opposing architecture is known as complex instruction set computing, i.e. CISC.

Various suggestions have been made regarding a precise definition of RISC, but the general concept is that of a system that uses a small, highly-optimized set of instructions, rather than a more specialized set of instructions often found in other types of architectures. Another common trait is that RISC systems use the load/store architecture, where memory is normally accessed only through specific instructions, rather than accessed as part of other instructions like add.

Although a number of systems from the 1960s and 70s have been identified as being forerunners of RISC, the modern version of the design dates to the 1980s. In particular, two projects at Stanford University and University of California, Berkeley are most associated with the popularization of the concept. Stanford's design would go on to be commercialized as the successful MIPS architecture, while Berkeley's RISC gave its name to the entire concept, commercialized as the SPARC. Another success from this era were IBM's efforts that eventually lead to the Power Architecture. As these projects matured, a wide variety of similar designs flourished in the late 1980s and especially the early 1990s, representing a major force in the Unix workstation market as well as embedded in laser printers, routers and similar products.

Well-known RISC families include DEC Alpha, AMD 29k, ARC, ARM, Atmel AVR, Blackfin, Intel i860 and i960, MIPS, Motorola 88000, PA-RISC, Power (including PowerPC), SuperH, and SPARC. In the 21st century, the use of ARM architecture processors in smart phones and tablet computers such as the iPad and Android tablets provided a wide user base for RISC-based systems. RISC processors are also used in supercomputers such as the K computer, the fastest on the TOP500 list in 2011, and the second at the 2012 list.

4.5 Virtual Memory

A cache stores a subset of the addresses space of RAM. An **address space** is the set of valid addresses. Thus, for each address in cache, there is a corresponding address in RAM. This subset of addresses (and corresponding copy of data) changes over time, based on the behavior of your program.

Cache is used to keep the most commonly used sections of RAM in the cache, where it can be accessed quickly. This is necessary because CPU speeds increase much faster than speed of memory access. If we could access RAM at 3 GHz, there wouldn't be any need for cache, because RAM could keep up. Because it can't keep up, we use cache.

What if we wanted more RAM than we had available. For example, we might have 1 M of RAM, what if we wanted 10 M? How could we manage?

One way to extend the amount of memory accessible by a program is to use disk. Thus, we can use 10 Megs of disk space. At any time, only 1 Meg resides in RAM.

In effect, RAM acts like cache for disk.

This idea of extending memory is called *virtual memory*. It's called "virtual" only because it's not RAM. It doesn't mean it's fake.

The real problem with disk is that it's really, really slow to access. If registers can be accessed in 1 nanosecond and cache in 5 ns and RAM in about 100 ns, then disk is accessed in fractions of seconds. It can be a million times slower to access disk than a register.

The advantage of disk is it's easy to get lots of disk space for a small cost.

Still, because disk is so slow to access, we want to avoid accessing disk unnecessarily.

Uses of Virtual Memory

Virtual memory is an old concept. Before computers had cache, they had virtual memory. For a long time, virtual memory only appeared on mainframes. Personal computers in the 1980s did not use virtual memory. In fact, many good ideas that were in common use in the UNIX operating systems didn't appear until the mid-1990s in personal computer operating systems (pre-emptive multitasking and virtual memory).

Initially, virtual memory meant the idea of using disk to extend RAM. Programs wouldn't have to care whether the memory was "real" memory (i.e., RAM) or disk. The operating system and hardware would figure that out.

Later on, virtual memory was used as a means of *memory protection*. Every program uses a range of addressed called the *address space*.

The assumption of operating systems developers is that any user program cannot be trusted. User programs will try to destroy themselves, other user programs, and the operating system itself. That seems like such a negative view, however, it's how operating systems are designed. It's not necessary that programs have to be deliberately malicious. Programs can be accidentally malicious (modify the data of a pointer pointing to garbage memory).

Virtual memory can help there too. It can help prevent programs from interfering with other programs.

Occasionally, you want programs to cooperate, and share memory. Virtual memory can also help in that respect.

How Virtual Memory Works

When a computer is running, many programs are simultaneously sharing the CPU. Each running program, plus the data structures needed to manage it, is called a *process*.

Each process is allocated an address space. This is a set of valid addresses that can be used. This address space can be changed dynamically. For example, the program might request additional memory (from dynamic memory allocation) from the operating system.

If a process tries to access an address that is not part of its address space, an error occurs, and the operating system takes over, usually killing the process (core dumps, etc).

How does virtual memory play a role? As you run a program, it generates addresses. Addresses are generated (for RISC machines) in one of three ways:

- A **load** instruction
- A **store** instruction
- Fetching an instruction

Load/store create data addresses, while fetching an instruction creates instruction addresses. Of course, RAM doesn't distinguish between the two kinds of addresses. It just sees it as an address.

Each address generated by a program is considered *virtual*. It must be translated to a real *physical address*. Thus, address translation is occurring all the time. As you might imagine, this must be handled in hardware, if it's to be done efficiently.

You might think translating each address from virtual to physical is a crazy idea, because of how slow it is. However, you get memory protection from address translation, so it's worth the hardware needed to get memory protection.

Paging

In a cache, we fetched quantities called data blocks or cache lines. Those are typically somewhere between, say, 4 and 64 bytes.

There is a corresponding terminology in virtual memory to a cache line. It's called a *page*.

A page is a sequence of N bytes where N is a power of 2.

These days, page sizes are at least 4K in size and maybe as large as 64 K or more.

Let's assume that we have 1M of RAM. RAM is also called physical memory. We can subdivide the RAM into 4K pages. Thus 1M / 4K = 256 pages. Thus, our RAM has 256 physical pages, each holding 4K.

Let's assume we have 10 M of disk. Thus, we have 2560 disk pages.

In principle, each program may have up to 4 G of address space. Thus, it can, in principle, access 2^{20} virtual pages. In reality, many of those pages are considered invalid pages.

Page Tables

How is an address translated from virtual to physical? First, like the cache, we split up a 32 bit virtual address into a virtual page (which is like a tag) and a page offset.

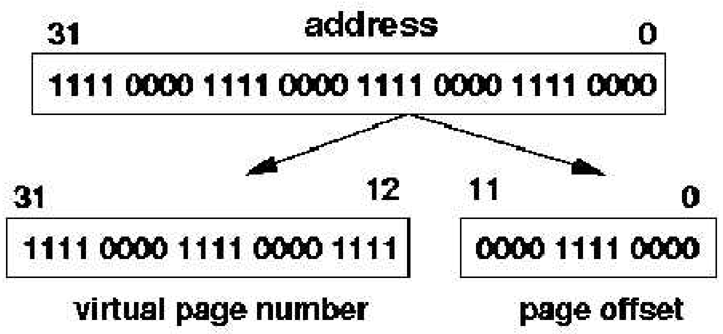


Figure 4-3

If this looks a lot like a fully-associative cache, but whose offset is much larger, it's because that's basically what it is.

We must convert the virtual page number to a physical page number. In our example, the virtual page consists of 20 bits. A page table is a data structure which consists of 2^{20} page table entries (PTEs). Think of the page table as an array of page table entries, indexed by the virtual page number.

The page table's index starts at 0, and ends at 2^{20} - 1.

Here's how it looks:

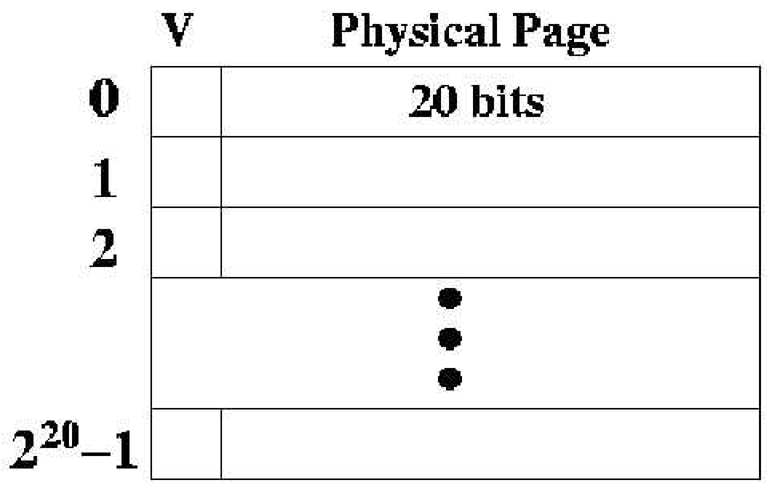


Figure 4-4

Suppose your program generates a virtual address. You'd extract bits B_{31-12} to get the virtual page number. Use that as an index into the above page table to access the page table entry (PTE).

Each PTE consists of a valid bit and a 20 bit physical page (it's 20 bits, because we assume we have 1M of RAM, and 1M of RAM requires 20 bits to access each byte). If the valid bit is 1, then the virtual page is in RAM, and you can get the physical page from the PTE. This is called a *page hit*, and is basically the same as a cache hit.

If the valid bit is 0, the page is not in RAM, and the 20 bit physical page is meaningless. This means, we must get the disk page corresponding to the virtual page from disk and place it into a page in RAM. This is called a *page fault*.

Because disk access is slow, slow, slow, we want to minimize the number of page faults.

In general, this is done by making RAM fully associative. That is, any disk page can go into any RAM page (disk, RAM, and virtual pages all have the same size).

In practice, some pages in RAM are reserved for the operating system to make the OS run efficiently.

Translation

Suppose your program generated the following virtual address $F0F0F0F0_{hex}$ (which is 1111 0000 1111 0000 1111 0000 1111 0000 $_{two}$). How would you translate this to a physical address? First, you would split the address into a virtual page, and a page offset (see below).

Then, you'd see if the virtual page had a corresponding physical page in RAM using the page table. If the valid bit of the PTE is 1, then you'd translate the virtual page to a physical page, and append the page offset. That would give you a physical address in RAM.

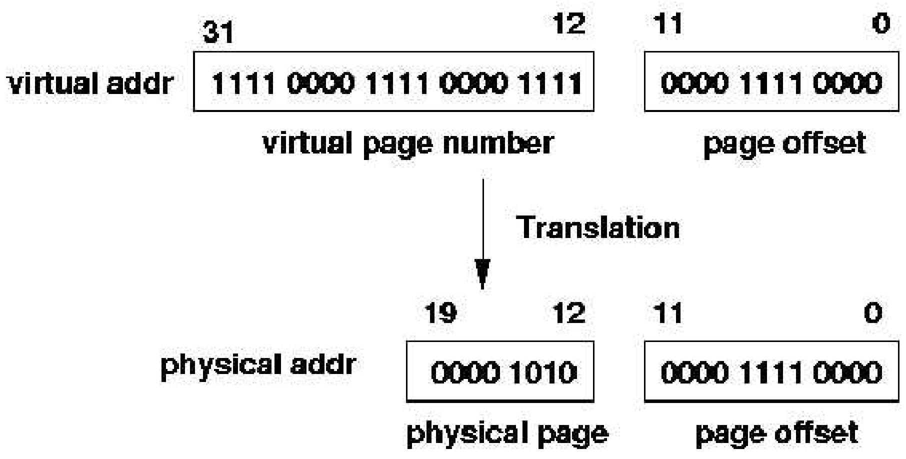


Figure 4-5

Huge Page Tables

Page tables can be very large. If every virtual page was valid, our page table would be 2^{20} X 21 bits. This is about 3 Megs just for one program's page table. If there are many programs, there are many tables, each occupying a lot of memory.

What's worse, the page tables we've been talking about are incomplete. If we have a page fault, we need to find the page on disk. Where is it located? That information is kept in another page table, which is indexed by the virtual page (same as the page table we talked about), and tells you where on disk to find it. Then, we have to copy that page to RAM, and update the first page table. Thus, we need two page table tables!

These page tables are basically just data. Thus, they occupy memory as any data occupies memory. When we switch from one process to another, we need to load its page table in RAM for easy access. It's useful to keep it located in certain parts of RAM for just such a purpose. If RAM is suitable large, we can have several processes' page tables in RAM at the same time.

A *page table register* can hold the physical address of the page table that's currently active to get quick access. Still, these are large, and we may want to find ways to speed things up.

Inverted Page Tables

There are many schemes to reduce the size of a page table. One way is to use a hierarchy. Thus, we might have two layers of pages. Bits $B_{31\text{-}22}$ might tell you the first layer, while $B_{21\text{-}13}$ might tell you the second layer.

Another idea is to use a kind of closed hash table. The hash table's size is based on the number of physical pages. The number of physical pages is usually a lot smaller than the number of all virtual pages put together.

A hash function takes a virtual page number as input, and produces an index into the hash table as the result. Each entry of the hash table consists of a virtual page number and a physical page number. You check to see if the virtual page number matched, and if so, then you use the physical page.

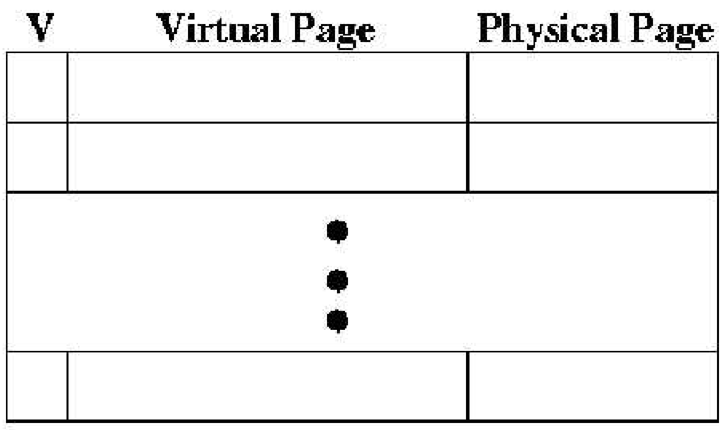
If it missed, then you must resolve the collision based on the hash table. In practice, you may need the number of entries of the hash table to be a few times larger than the number of physical pages, to avoid excessive collisions.

An inverted page table takes longer to access because you may have collisions, but it takes up a lot less memory. It helps with page hits. However, if you have a page fault, you still need a page table that maps virtual pages to disk pages, and that will be large.

Translation Look aside Buffer (TLB)

What's the cost of address translation? For each virtual address, we must access the page table to find the PTE corresponding to the virtual page. We look up the physical page from the PTE, and construct a physical address. Then, we access RAM at the physical address. That's two memory acccess: one to access the PTE, one more to access the data in RAM. Cache helps us cut down the amount of time to access memory, but that's only if we have cache hits.

The idea of a TLB is to create a special cache for translations. Here's one example of a TLB.



Translation Lookaside Buffer (TLB)

Figure 4-6

Each row in the TLB is like one slot of a cache. Assume we have 64 rows. When you have a virtual address, you can split it into a virtual page and an offset.

In parallel, compare the virtual page to all of the entries of the TLB (say, 64). There should be, at most, one match. Just like a fully associative cache, you want to check if the TLB entry is valid.

If a TLB hit occurs, replace the virtual page with a physical page to create a physical address.

If there's a TLB miss, then it's still possible that the virtual page resides in RAM. You must now look up the PTE (page table entry) to see if this is the case. If the PTE says the virtual page is in RAM, then you can update the TLB, so that it has a correct virtual to physical page translation.

The TLB is designed to only store a limited subset of virtual to physical page translation. It is really just a cache for the page table, storing only the most frequently used translations.

The TLB can be kept small enough that it can be fully associative. However, some CPU designers make larger TLBs that are direct mapped or set associative.

Memory Protection

How does a virtual address give us memory protection? Suppose you work at a post office, which assigns post boxes to individuals. You have a person who comes in and says they want three post office boxes: 100, 101, and 102.

They insist on using those numbers. Another customer comes in, and insists on using those numbers too. How can you accommodate both customers?

Basically, you cheat. You tell the first customer you have boxes 100, 101, and 102, but you assign him boxes 200, 201, and 202. Similarly, you tell the second customer that you also have boxes 100, 101, and 102, but you assign her boxes 320, 321, and 322.

Whenever customer 1 wants the mail in box 100, you translate it to box 200. Whenever customer 2 wants mail in box 100, you translate it to box 320. Thus, your two customers get to use the box numbers they want, and through the magic of translation, they two customers avoid using each other's boxes.

If the post office wanted to reserve its own boxes for its own use, it could reserve boxes 1 through 100 to itself, and never assign those boxes, directly or indirectly to a customer. Even if a customer wants box 50, they can be assigned box 150, safely outside the range of boxes reserved for the post office.

This same analogy applies to real programs. Each program can assume it uses the same set of 32 bit virtual addresses. We just make sure that those virtual pages do not map to the same disk page, nor to the same physical page.

Who's job is it to assign the pages? It's the operating system. When a program starts up, it will want a certain range of addresses. The operating system creates a page table for the program, making sure the disk pages it uses do not conflict with the disk pages of other programs.

Invalid Pages

Sometimes you don't really want a program to access all possible 32 bit addresses. This helps reduce the total size of the page table. One way to prevent a user program from accessing invalid pages is making certain virtual pages entries invalid. Thus, an attempt to translate virtual to physical page will fail, and even looking up the virtual page on disk fails.

Page Replacement Schemes

Like cache, you can have page replacement schemes based on FIFO, LRU, LFU, etc. In general, page replacement schemes can be more sophisticated because getting a page off disk is really slow, so you can afford to take more time to make a better choice.

Dirty Bit

In reality, caches usually don't have dirty bits. This means that you must always write back if a cache line is evicted.

However, because disk access is slower, it makes sense to use dirty bits for pages. Thus, if a page hasn't been modified (maybe because it's read only), there's no reason to copy it back to disk. Just toss it out.

Cache

Virtual memory works with caching. Basically, once the virtual address is translated to a physical address, and then the physical address is passed to the cache, which checks to see if there is a cache hit.

The Oblivious Programmer

As with cache, assembly language programmers don't have to worry about virtual memory. They just see "memory". They don't do anything in particular whether there's virtual memory or not. Virtual memory is handled partly by hardware (translation mechanism) and partly by the operating system (sets up page table, handles page faults, etc).

This is good because at one point, programmers had to worry very much about whether a chunk of memory resided on the disk or in RAM. Programmers had to spend a great deal of effort managing this, and it distracted them from coding. With virtual memory, the management of disk as an extension of RAM is handled automatically.

Shared Memory

You can share memory between two processes by mapping the virtual page to the same disk page. When that disk page is resident in physical memory, then both processes can access the same location.

There may be issues of synchronization to handle, but that's a topic that's best left to a course in operating systems. Suffice it to say that we do have a way to map virtual pages to the same disk page to allow for sharing.

Sharing is available when you want two processes to collaborate in a somewhat safe manner. Both processes, for the most part, have their own memory. They only share a small region between the two of them.

Terminology

Some definitions before we summarize:

- A physical page is 2^k consecutive bytes in memory. The page is "super word-aligned". That is, it's first byte must be at an address divisible by 2^k.
- A disk page is 2^k consecutive bytes in disk (also, super word-aligned) Disks can be partitioned up non-contiguously, so it doesn't have to be truly consecutive, as long as the CPU "thinks" it is. Each physical page should have a corresponding disk page. However, there are usually far more disk pages than physical pages.
- A virtual page is 2^k consecutive bytes, which is super word-aligned. This is the page the program thinks it is dealing with. This page can either be in RAM (physical memory) and/or disk.

The physical page number, disk page number, and virtual page number can all be different. The virtual page number is the page used by the program. The physical page number is the page in RAM, if it is currently in RAM. The disk page number is the page in disk. Recall the post office analogy where the person has a virtual post office box number, and the postal workers know the mapping for each person from virtual box number to real box number.

Check your progress

Q1. Define RISC.

Q2. What is virtual memory? Explain.

Q3. Define Paging, Page table and TLB.

4.6 Summary

In this unit you learnt about Interrupt Structures, Multiprogramming, Processor Features, Reduced Instruction Set Computers (RISC), Virtual memory

- Interrupt structure refers to the precedence of interrupts.
- Superscalar architecture usually is associated with high-output Reduced Instruction Set Computer (RISC) chips.
- MMX technology was originally named for multimedia extensions, or matrix math extensions,
- It's also possible to have memory protection without any disks.
- A programmer might be able to access all disk pages, if the operating system allowed it when setting up page tables.

4.7 Review Questions

Q1. What do you mean by interrupt structures? Explain with example.

Q2. What is a multiprogramming? Explain in detail.

Q3. Define the processor features?

Q4. Write a short note on Reduced Instruction Set Computers.

Q5. Define virtual memory in detail.

BIBLIOGRAPHY

Amdahl, G.M., *et al* (1964). Architecture of IBM system/360. *IBM Journal of Research and Development*, 8(2), 87-101.

Bradley, A.C. (1989). *Optical storage for computers: technology and applications*.Chichester: Ellis Horwood Limited.

Ceruzzi, P.E. (1998). *A history of modern computing*. Cambridge, M.A.: MIT Press.

Clemens, A. (2000). The undergraduate curriculum in computer architecture. *IEE Micro*, May /June 2000.

Halal, William E. (1993). The information technology revolution: computer hardware, software and services into the 21st century. *Technological Forecasting and Social Change*, 44, 69-86.

Hennessy, J.L. and Patterson, D.A. (1990). *Computer architecture: a quantitativeapproach*. N.P: Morgan-Kauffman.

Hussain, Khateeb M. and Hussain, Donna (1989). *Computers: technology applications, and social implications*. New Delhi: Prentice-Hall of India.

Hwang, K. (1993). *Advanced computer architecture*. New York: Mc Graw-Hill.

Martin, James (1989). *Information engineering: introduction*. New Jersey: Prentice- Hall.

Martin, William J. (1995). *Global information society*. 2nd rev.ed. London: Aslib.pp. 33-63.

Saffady, William. (1992). *Optical storage technology 1992: a state of the art review*. Westport: Meckler.

Sima, Dezso, *et al* (2003). *Advanced computer architectures: a design space*

Approach. Delhi: Pearson Education Inc.

Smith, R.E. (1989). A historical overview of computer architecture. *Annals of the History of Computing*, 10, 277-303.

Stallings, William. (2003). *Computer organisation and architecture: designing for performance*. 6th ed. Delhi: Pearson Education Inc.

Vranesic, Z. and Thurber, K. (1980). Teaching computer structures. *Computer*, June 1980.

Uttar Pradesh Rajarshi Tandon
Open University

MBA -3.51
Computer Fundamentals and its Organisation

BLOCK

4

Concepts of Software & Operating System

Unit-I	**223**
Software Concepts	
Unit-II	**229**
Software Qualities	
Unit-III	**246**
Operating System	
Unit-IV	**256**
Operating System Types	

Printed and Published by Prof. P.P. Dubey, Registrar, Uttar Pradesh Rajarshi Tandon Open University , Prayagraj, 2022.

Printed By K.C.Printing & Allied Works, Panchwati, Mathura -281003.

Block - 4 : Concepts of Software & Operating System

This is the fourth block of this course and focused on the concepts of software & operating system. This block has detailed description of software and its types. It also focused on operating system and having four following units.

In the first unit of this block we introduce the concept of software, types of software like system and application software. We also discussed about the different programming languages like algorithmic languages, business-oriented languages, education-oriented languages, object-oriented languages, declarative languages, scripting languages, document formatting languages, World Wide Web display languages.

In the second unit we focused on nature of software, qualities of software, and different views of software quality. These views are transcendental view, user view, manufacturing view, product view, and value-based view and in the last we described about the business value of quality. .

In the third unit we discussed about the operating systems its history and evolution. This unit also covers the functions of operating system. These functions are process management, memory management, device management, file management, security management, and command interpretation. We also discussed the concept of multitasking and its functionality.

In the last unit we told about the multiprocessing, difference between multiprogramming and multiprocessing, advantages and limitations of multiprocessing. We also covered the concept of time-sharing, requirements of time-sharing systems, advantages of time-sharing systems, real-time operating system, difference: RTOS v/s general purpose OS, RTOS classification.

As you study the material you will come across abbreviations in the text, e.g. Sec. 1.1, Eq.(1 .1) etc. The abbreviation Sec. stands for section, and Eq. for equation. Figure, a. b refers to the bth figure of Unit a, i.e. Figure. 1.1 is the first figure in Unit 1. Similarly, Sec. 1.1 is the first section in Unit 1 and Eq.($.8) is the eighth equation in Unit 4. Similarly Table x. y refers to the yth table of Unit x, i.e. Table. 1.1 is the first table in Unit 1.

In your study you will also find that the units are not of equal-length and your study time for each unit will vary.

We hope you enjoy studying the material and wish you success.

UNIT-I

Software Concepts

Structure

1.0 Introduction

1.1 Software

1.2 Types of Software

1.3 Programming Languages

1.4 Summary

1.5 Review Questions

1.0 Introduction

This is the first unit of this block. In this unit we focused on software concepts. There are five sections in this unit. In Sec. 1.1 you will know about software. Sometimes abbreviated as SW and S/W, software is a collection of instructions that enable the user to interact with a computer, its hardware, or perform tasks. Without software, computers would be useless. For example, without your Internet browser, you could not surf the Internet or read this page and without an operating system, the browser could not run on your computer. The picture to the right shows a Microsoft Excel box, an example of a spreadsheet software program. In Sec. 1.2 we discussed about types of software. There are two main types of software: systems software and application software. Systems software includes the programs that are dedicated to managing the computer itself, such as the operating system, file management utilities, and disk operating system (or DOS). In Sec. 1.3 you will learn about programming language. A programming language is a formal computer language or constructed language designed to communicate instructions to a machine, particularly a computer. Programming languages can be used to create programs to control the behavior of a machine or to express algorithms. In Sec. 1.4 and 1.5 you will find summary and review questions respectively.

Objectives

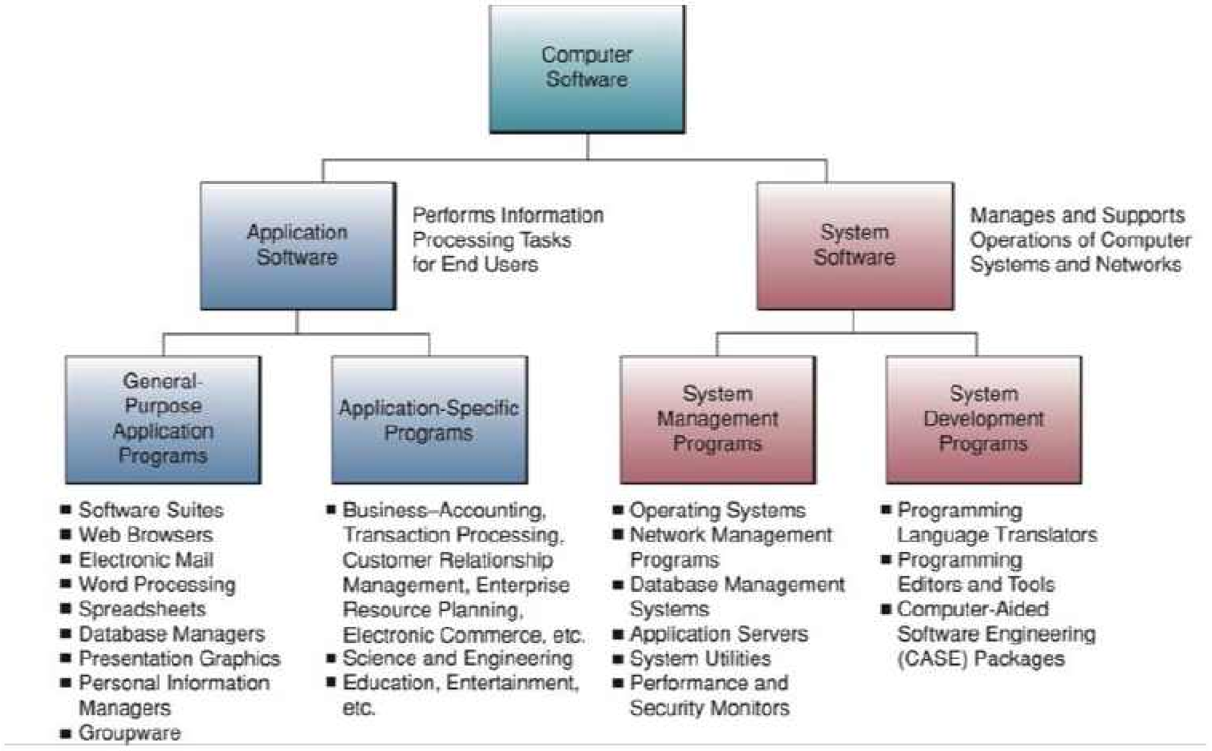
After studying this unit you should be able to:

- Define Software & its types
- Describe Programming Languages

1.1 Software

Computer software, or just **software**, is any set of machine-readable instructions (most often in the form of a computer program) that directs a computer's processor to perform specific operations. The term is used to contrast with computer hardware, the physical objects (processor and related devices) that carry out the instructions. Hardware and software require each other; neither has any value without the other.

1.2 Types of Software



Computer Software
Application Software
Performs Information Processing Tasks for End Users
System Software
Manages and Supports Operations of Computer Systems and Networks
General-Purpose Application Programs
Application-Specific Programs
System Management Programs
System Development Programs
Software Suites
Web Browsers
Electronic Mail
Word Processing
Spreadsheets
Database Managers
Presentation Graphics
Personal Information Managers
Groupware
Business–Accounting, Transaction Processing, Customer Relationship Management, Enterprise Resource Planning, Electronic Commerce, etc.
Science and Engineering
Education, Entertainment, etc.
Operating Systems
Network Management Programs
Database Management Systems
Application Servers
System Utilities
Performance and Security Monitors
Programming Language Translators
Programming Editors and Tools
Computer-Aided Software Engineering (CASE) Packages

Figure 1-1 Software Types

System Software

System software is a program that manages and supports the computer resources and operations of a computer system while it executes various tasks such as processing data and information, controlling hardware components, and allowing users to use application software. That is, systems software functions as a *bridge* between computer system hardware and the application software. System software is made up of many control programs, including the operating system, communications software and database manager. There are many kinds of computers these days. Some of them are easier to learn than others. Some of them perform better than others. These differences may come from different systems software.

Three Kinds of Programs

Systems software consists of three kinds of programs. The system management programs, system support programs, and system development programs are they. These are explained briefly.

System Management Programs

These are programs that manage the application software, computer hardware, and data resources of the computer system. These programs include operating systems, operating environment programs, database management programs, and telecommunications monitor programs. Among these, the most important system management programs are operating systems. The operating systems are needed to study more details. There are two reasons. First, users need to know their functions first. For the second, there are many kinds of operating systems available today.

Telecommunications monitor programs are additions of the operating systems of microcomputers. These programs provide the extra logic for the computer system to control a class of communications devices.

System Support Programs

These are the programs that help the operations and management of a computer system. They provide a variety of support services to let the computer hardware and other system programs run efficiently. The major system support programs are system utility programs, system performance monitor programs, and system security monitor programs (virus checking programs).

System Development Programs

These are programs that help users develop information system programs and prepare user programs for computer processing. These programs may analyze and design systems and program itself. The main system development programs are programming language translators, programming environment programs, computer-aided software engineering packages.

APPLICATION SOFTWARE

Application software consists of Programs that direct computers to perform specific information processing activities for end users. These programs are called *application* packages because they direct the processing required for a particular use, or *application*, which users want to accomplish. Thousands of application packages are available because there are thousands of different jobs end users want computers to do.

Kinds of Application Software

Application software includes a variety of programs that can be subdivided into general-purpose and application-specific categories.

General-Purpose Application Programs

General-purpose applications packages are programs that perform common information processing hobs for end users. For example, word processing programs, electronic spreadsheet programs, database management programs, graphics programs, communications programs, and integrated packages are popular with microcomputer users for home, education, business, scientific, and many other general purposes.

They are also known as *productivity packages*, because they significantly increase the productivity of end users. This packaged software is also called *off-the-shelf software packages*, because these products are packaged and available for sale. Many features are common to most packaged programs.

Application-Specific Software

Many application programs are available to support specific applications of end users. *Business Application Programs*: Programs that accomplish the information processing tasks of important business functions or industry requirements.

Scientific Application Programs: Programs that perform information processing tasks for the natural, physical, social, and behavioral sciences, engineering and all other areas involved in scientific research, experimentation, and development. There are so many other application areas such as education, music, art, medicine, etc.

Application Software Trends

The trend in computer application software is toward multipurpose, expert-assisted packages with natural language and graphical user interfaces. There are two major trends:

Off-The-Shelf Software Packages

There is a trend away from custom-designed one-of- a-kind programs developed by the professional programmers or end users of an organization.

Instead, the trend is toward the use of the *"off-the-self"* software package acquired by end users from software vendors. This trend accelerated with the development of *inexpensive* and *easy-to-use* productivity software packages for microcomputers, and it continues to grow.

Nonprocedural, Natural Languages

There is a major trend away from technical, machine-specific programming languages using binary-based or symbolic codes and from procedural languages, which use English-like statements and mathematical expressions to specify the sequence of instructions a computer must perform. Instead, the trend is toward nonprocedural, natural languages that are closer to human conversation. This trend has accelerated with the creation of easy-to-use, nonprocedural *fourth- generation languages* (4GL). It continues to grow as developments in graphics and artificial intelligence produce natural language and *graphical interfaces* that make software packages easier to use.

POPULAR APPLICATIONS SOFTWARE

Word Processing Packages

Word processing software is used to create, manipulate, and print documents. Documents can be any kind of text material. Some examples of documents are letters, memos, term papers, reports, and contracts.

The beauty of the computer word processor is that users can make any changes or corrections before printing out the document. Even after users document is printed out, users can easily go back and make changes. Users can then print it out again. Popular word processing packages include WordPerfect, MS-Word, and MacWrite. These word processing packages allow users to do the following interesting features:

- **Word Wrap/Enter Key:** One outstanding word processing feature is a word wrap. A word processor decides for Users and automatically moves the cursor to the next line. As users keep typing, the words "wrap around" to the next line. To begin a new paragraph or leave a blank line, users press the Enter key.
- **Search/Replace:** A search command allows users to find any number that users know exists in users' document. When users search, the cursor will move to the first place where the item appears. The replace command automatically replaces the word users search for with another word. The search and replace commands are useful for finding and fixing errors.
- **Block/Move:** The portion of text users wish to move is a block. User's mark the block by giving commands that produce highlighting, a band of light over the area. The task of moving the block is called a block move. The block command may also be used to delete text or to copy chunks of text into another document.
- **Margins:** Margins may be justified to right, left or full (right and left at the same time) in the most word processing packages. That is, they may be evened up to the right, left or both side simultaneously.
- **Centering/Emphasizing:** Headings of a document may be centered. Words or phrases may be typed underlined or boldface (extra dark lettering) for emphasis.
- **Spelling Checker:** A spelling- checker program can check spelling errors in a document automatically.
- **Thesaurus:** This programs enable users to quickly find the right word or an alternative word by presenting users with an on-screen thesaurus.

- **Mail Merge :** This feature allows users to merge different names and addresses so that users can mail out the same form letter to different people.
- **Desktop Publishing :** Today's advanced word processing programs can perform desktop publishing capabilities. This feature enables users to mix text and graphics to produce newsletters and other publications of nearly professional quality.
- **Outliner:** Sometimes called *idea processors*. It helps users organize and outline users' thoughts before users prepare a document or develop a presentation.
- **Grammar/Style Checker :** These programs can be used to identify and correct grammar and punctuation errors.
- **Importing :** Most of the programs have an *importing* feature. Files may be retrieved from non-text programs such as spreadsheets and graphics and added to the word processing program.

Electronic Spreadsheet Packages

A spreadsheet is an electronic worksheet used to organize and manipulate numbers and display options for what-if analysis. The electronic spreadsheet has rows and columns stored in the computer's memory and displayed on its video screen. Electronic spreadsheets allow users to try out various *what-if* kinds of possibilities. That is a powerful feature. Users can manipulate numbers by using stored formulas and calculate different outcomes. A spreadsheet has several parts. The *worksheet area* of the spreadsheet has *column headings* across the top and *row headings* down the left-hand side. The intersection of a column and row is called a *cell*. The cell holds a unit of information. The position of a cell is called the *cell address*. A *cell pointer* (spread sheet cursor) indicates where data is to be entered or changed in the spreadsheet.

Popular electronic spreadsheet packages include Lotus 1-2-3, Quattro Pro, and Excel. Some common features of spreadsheet programs are as follows:

- **Format:** Column and row headings are known as *labels*. Usually a label is a word or symbol. A number in a cell is called a *value*. A label can be centered in the cell or positioned to the left or right. A value can be displayed to show decimal places, dollars, or percent (%). The number of decimal positions (if any) can be altered, and the width of columns can be changed.
- **Formulas:** The major benefit of spreadsheets is that users can manipulate data by using formulas. They make connections between numbers in particular cells.
- **Recalculation:** Recalculation is the most important feature of spreadsheets. If users change one or more numbers in users' spreadsheet, all related formulas will recalculate automatically. By manipulating the values, users can use spreadsheet formulas to explore users' options.
- **Windows:** The screen-sized area of a spreadsheet that users can view is called a *window* or a *page*. Only about 20 rows and 8 columns of a spreadsheet are visible on the video display screen at one time. The total size of the spreadsheet can be much larger.
- **Graphic Data Display:** Most spreadsheets allow users to present their data in graphic form. That is, users can display numerical information as pie charts or bar charts.
- **3-D Graphics:** Most spreadsheet programs even permit users to display data in graphs and charts that have a three-dimensional look.

- **Graphics on Worksheet:** A new feature gives users the ability to place graphical elements such as lines, arrows, and boxes directly onto the worksheet. Users can create charts and graphs directly on the worksheet.
- **Consolidation Feature:** Data may be consolidated from several small worksheets into one large worksheet. Thus, users can work with small worksheets, which are more manageable, and summarize the data on a large worksheet.
- **Dynamic File Links:** Some software offers dynamic file links, which allow users to link cells in one worksheet file to cells in other worksheet files. Whenever a change occurs in one file, the linked cells in the other files are automatically updated.

Database Management Packages

A *database* is a large collection of data entered a computer system and stored for future use. The computerized information in the database is organized so that the parts that have something in common can be retrieved easily. Most DBMS packages can perform four primary tasks:

- **Database Development:** Define and organize the content, relationships, and structure of the data needed to build a database.
- **Database Interrogation:** Access the data in a database for information retrieval and report generation. A user can selectively retrieve and display information and produce printed reports and documents.
- **Database Maintenance:** Add, delete, update, correct, and protect the data in a database.
- **Application Development:** Develop prototypes of data entry screens, queries, forms, reports, and labels for a proposed application.

A database management package or *database management system* (*DBMS*) is a software package used to set up, or structure, a database. It is also used to retrieve information from a database. The top part of the figure is a *menu*. The entire list of member names and addresses is called a *file*. Each line of information about one member is called a *record*. Each column of information within a record is called a *field*.

Popular database management programs include dBASE, Paradox, and FoxPro. Database management packages have different features, depending on their sophistication. A principal feature of database management software for microcomputers is as follows:

- **Retrieve/Display:** A basic feature of all database programs is the capability to locate records in the file quickly. The program can search each record for a match in a particular field to whatever data users specify. The records can then be displayed on the screen for viewing, updating, or editing.
- **Sort:** Database management packages make it easy to change the order of records in a file. Normally, records are entered the database in the order they occur. There are many ways users can quickly rearrange the records in the file, such as by employees' last name or by their social security number.
- **Calculate/Format:** Many database programs contain built-in math formulas. In the office, for example, users can use this feature to find the highest or lowest commissions earned. Users can calculate the average of the commissions earned by the sales force in one part of the country. This information can be organized as a table and printed out in a report format.

- **Customized Data-Entry Forms:** A person new to the database program may find some descriptions for fields confusing. For example, a field name may appear as "CUSTNUM" for "customer number." However, the form on the screen may be customized so that the expression "Enter the customer number" appears for "CUSTOM." Fields may also be rearranged on the screen, and boxes and lines may be added.
- **Professional-Looking Reports:** A custom-report option enables users to design the elements users want in a report. Examples are the descriptions appearing above columns and the fields users wish to include. Users can even add graphic elements, such as a box or line, so that the printed report has a professional appearance.
- **Program Control Languages:** Most people using a database management program can accomplish everything they need to do by making choices from the menus. Many database management programs include a programming control language so that advanced users can create sophisticated applications.

Graphics Packages

A graphics program can display numeric data in a visual format for analytical or presentation purposes. Any other types of presentation graphics displays are possible. *Draw* and *Input* graphics packages support freehand drawing, while desktop publishing programs provide pre-drawn *clip art* graphics for insertion into documents. Popular business graphics packages are Harvard Graphics, Freelance, Corel Draw, and etc. There are two types of graphics programs. *Analytical graphics programs* are used to analyze data. *Presentation graphics programs* are used to create attractive finished graphs for presentations or reports.

- **Analytical Graphics:** Analytical graphics make numerical data much easier to grasp than when it is as rows and columns of numbers. Graphics may take the form of bar charts, line graphs, and pie charts. The bar chart gives an instant visual profile of the some figures. The line graph shows a visual profile in another way. The pie chart shows the proportion of some figures as slices of pie. High-low graph shows a range, such as house prices. Most analytical graphics programs come as part of spreadsheet programs, such as Lotus 1-2-3 and Quattro-Pro. They are helpful in displaying economic trends, sales figures, and the like for easy analysis. Analytical graphics may be viewed on a monitor or printed out.
- **Presentation Graphics:** Users can use presentation graphics to communicate a message or to persuade other people, such as supervisors or clients. Thus, presentation graphics are used by marketing or sales people, for example. Presentation graphics look more sophisticated than analytical graphics, using color, titles, a three- dimensional look, and other features a graphic artist might use. High-end presentation graphics packages even include animation capabilities. These packages allow users to create and edit animated graphics on user's microcomputer.

Communications Packages

Communications software packages for microcomputers are also viewed as *general-purpose* application packages. These packages can connect a microcomputer equipped with a *modem* to a public and private network. Communications software enables a microcomputer to send and receive data over a telephone or other communications line. Communications programs are used by all kinds of people inside and outside business. Examples are students doing research papers, travelers making plane reservations, consumers buying products, investors getting stock quotations, and economists getting government statistical data. Communications programs give microcomputers a powerful feature, which is connectivity.

Connections with microcomputers open a world of services. Popular communications software includes ProComm, Smartcom, and Crosstalk. Some common features of microcomputer communications programs are as follows:

- **Data Banks: With a communications program, users can access enormous computerized databases - data banks of information. Some of these, such as Dialog, resemble huge electronic encyclopedias.**
- **Message Exchanges: Communications programs enable users to leave and receive messages on electronic bulletin boards or to use electronic-mail services. Electronic bulletin boards exist for people interested in swapping all kinds of software or information.**
  Many organizations now have electronic mailboxes. For instance, users can transmit a report users have created on users' word processor to a faraway company executive or to a college instructor.
- **Financial Services:** With communications programs, users can look up airline reservations and stock quotations. Users can order discount merchandise and even do home banking and bill paying.

Integrated Packages

Integrated packages combine the abilities of several general-purpose applications in one program. Integrated software is an all-in-one application package that includes word processing, spreadsheet, database manager, graphics, and communications. An integrated package works together and shares information from one program with another. Integrated packages were developed to solve the problems caused by the inability of individual programs to communicate and work with common files of data.

Some integrated packages require significant amounts of memory and may compromise on the speed, power, and flexibility of some of their functions to achieve integration. Powerful microcomputers available these days, however, allow users to accomplish all their works without sacrificing computers' speed and flexibility.

What happens if users want to take the data in one program and use it in another? Suppose users want to take information stored in the database manager and use it in a spreadsheet. This is not always possible with separate application packages, but it is with integrated software. With an integrated package, users can use the database manager to pull together relevant facts. An example of such facts might be the annual membership fees for a sports club for different years. Users can then use the spreadsheet to compare these membership fees. Users can use the word processing program to write a memo about these membership fees for different categories of members. Users can use this program to merge into the memo totals from the spreadsheet. Finally, users can use the communications program to send the memo to another computer. Some popular integrated packages are Works, First Choice, Symphony, Enable, Framework, Smart Ware 11, Microsoft Office, and Perfect Office. End users who are just beginning to learn about application software find integrated packages quite helpful. These packages can easily exchange data between programs, and they share a common structure. These factors make them easy to learn and convenient to use.

Check your progress

Q1. What do you mean by Software?

Q2. Define the types of software?

1.3 Programming Languages

Computer programming language, any of various languages for expressing a set of detailed instructions for a digital computer, Such instructions can be executed directly when they are in the computer manufacturer-specific numerical form known as machine language, after a simple substitution process when expressed in a corresponding assembly language, or after translation from some "higher-level" language. Although there are over 2,000 computer languages, relatively few are widely used.

Machine and assembly languages are "low-level," requiring a programmer to manage explicitly all of a computer's idiosyncratic features of data storage and operation. In contrast, high-level languages shield a programmer from worrying about such considerations and provide a notation that is more easily written and read by programmers.

Language types

Machine and assembly languages

A machine language consists of the numeric codes for the operations that a particular computer can execute directly. The codes are strings of 0s and 1s, or binary digits ("bits"), which are frequently converted both from and to hexadecimal (base 16) for human viewing and modification. Machine language instructions typically use some bits to represent operations, such as addition, and some to represent operands, or perhaps the location of the next instruction. Machine language is difficult to read and write, since it does not resemble conventional mathematical notation or human language, and its codes vary from computer to computer.

Assembly language is one level above machine language. It uses short mnemonic codes for instructions and allows the programmer to introduce names for blocks of memory that hold data. One might thus write "add pay, total" (Assembly Language) instead of "0110101100101000" (Machine Language) for an instruction that adds two numbers.

Assembly language is designed to be easily translated into machine language. Although blocks of data may be referred to by name instead of by their machine addresses, assembly language does not provide more sophisticated means of organizing complex information. Like machine language, assembly language requires detailed knowledge of internal computer architecture. It is useful when such details are important, as in programming a computer to interact with input/output devices (printers, scanners, storage devices, and so forth).

Algorithmic languages

Algorithmic languages are designed to express mathematical or symbolic computations. They can express algebraic operations in notation similar to mathematics and allow the use of subprograms that package commonly used operations for reuse. They were the first high-level languages.

FORTRAN

The first important algorithmic language was FORTRAN (*for*mula *tran*slation), designed in 1957 by an IBM team led by John Backus. It was intended for scientific computations with real numbers and collections of them organized as one- or multidimensional arrays. Its control structures included conditional IF statements, repetitive loops (so-called DO loops), and a GOTO statement that allowed non sequential execution of program code. FORTRAN made it convenient to have subprograms for common mathematical operations, and built libraries of them.

FORTRAN was also designed to translate into efficient machine language. It was immediately successful and continues to evolve.

ALGOL

ALGOL (*algo*rithmic *l*anguage) was designed by a committee of American and European computer scientists during 1958–60 for publishing algorithms, as well as for doing computations. Like LISP, ALGOL had recursive subprograms—procedures that could invoke themselves to solve a problem by reducing it to a smaller problem of the same kind. ALGOL introduced block structure, in which a program is composed of blocks that might contain both data and instructions and have the same structure as an entire program. Block structure became a powerful tool for building large programs out of small components.

ALGOL contributed a notation for describing the structure of a programming language, Backus–Naur Form, which in some variation became the standard tool for stating the syntax (grammar) of programming languages. ALGOL was widely used in Europe, and for many years it remained the language in which computer algorithms were published. Many important languages, such as Pascal and Ada (both described later), are its descendants.

LISP

LISP (*list p*rocessing) was developed about 1960 by John McCarthy at the Massachusetts Institute of Technology (MIT) and was founded on the mathematical theory of recursive functions (in which a function appears in its own definition). A LISP program is a function applied to data, rather than being a sequence of procedural steps as in FORTRAN and ALGOL. LISP uses a very simple notation in which operations and their operands are given in a parenthesized list. For example, (+ *a* (* *b c*)) stands for *a* +*b***c*. Although this appears awkward, the notation works well for computers. LISP also uses the list structure to represent data, and, because programs and data use the same structure, it is easy for a LISP program to operate on other programs as data.

LISP became a common language for artificial intelligence (AI) programming, partly owing to the confluence of LISP and AI work at MIT and partly because AI programs capable of "learning" could be written in LISP as self-modifying programs. LISP has evolved through numerous dialects, such as Scheme and Common LISP.

C

The C programming language was developed in 1972 by Dennis Ritchie and Brian Kernighan at the AT&T Corporation for programming computer operating systems. Its capacity to structure data and programs through the composition of smaller units is comparable to that of ALGOL. It uses a compact notation and provides the programmer with the ability to operate with the addresses of data as well as with their values. This ability is important in systems programming, and C shares with assembly language the power to exploit all the features of a computer's internal architecture. C++, along with its descendant C++++, remains one of the most common languages.

Business-oriented languages

COBOL

COBOL (*co*mmon *b*usiness *o*riented *l*anguage) has been heavily used by businesses since its inception in 1959. A committee of computer manufacturers and users and U.S. government organizations established CODASYL (*Co*mmittee on *Da*ta *Sy*stems and *L*anguages) to develop and oversee the language standard in order to ensure its portability across diverse systems.

COBOL uses an English-like notation—novel when introduced. Business computations organize and manipulate large quantities of data, and COBOL introduced the record data structure for such tasks. A record clusters heterogeneous data such as a name, ID number, age, and address into a single unit. This contrasts with scientific languages, in which homogeneous arrays of numbers are common.

Records are an important example of "chunking" data into a single object, and they appear in nearly all modern languages.

SQL

SQL (structured query language) is a language for specifying the organization of databases (collections of records). Databases organized with SQL are called relational because SQL provides the ability to query a database for information that falls in a given relation. For example, a query might be "find all records with both last_name Smith and city New York." Commercial database programs commonly use a SQL-like language for their queries.

Education-oriented languages

BASIC

BASIC (beginner's all-purpose symbolic instruction code) was designed at Dartmouth College in the mid-1960s by John Kemeny and Thomas Kurtz. It was intended to be easy to learn by novices, particularly non-computer science majors, and to run well on a time-sharing computer with many users. It had simple data structures and notation and it was interpreted: a BASIC program was translated line-by-line and executed as it was translated, which made it easy to locate programming errors.

Its small size and simplicity also made BASIC a popular language for early personal computers. Its recent forms have adopted many of the data and control structures of other contemporary languages, which makes it more powerful but less convenient for beginners.

PASCAL

About 1970 Niklaus Wirth of Switzerland designed Pascal to teach structured programming, which emphasized the orderly use of conditional and loop control structures without GOTO statements. Although Pascal resembled ALGOL in notation, it provided the ability to define data types with which to organize complex information, a feature beyond the capabilities of ALGOL as well as FORTRAN and COBOL. User-defined data types allowed the programmer to introduce names for complex data, which the language translator could then check for correct usage before running a program.

During the late 1970s and '80s, Pascal was one of the most widely used languages for programming instruction. It was available on nearly all computers, and, because of its familiarity, clarity, and security, it was used for production software as well as for education.

LOGO

Logo originated in the late 1960s as a simplified LISP dialect for education; Seymour Papert and others used it at MIT to teach mathematical thinking to schoolchildren. It had a more conventional syntax than LISP and featured "turtle graphics," a simple method for generating computer graphics. (The name came from an early project to program a turtle like robot.) Turtle graphics used body-centered instructions, in which an object was moved around a screen by commands, such as "left 90" and "forward," that specified actions relative to the current position and orientation of the object rather than in terms of a fixed framework. Together with recursive routines, this technique made it easy to program intricate and attractive patterns.

HYPERTALK

Hypertalk was designed as "programming for the rest of us" by Bill Atkinson for Apple's Macintosh. Using a simple English-like syntax, Hypertalk enabled anyone to combine text, graphics, and audio quickly into "linked stacks" that could be navigated by clicking with a mouse on standard buttons supplied by the program. Hypertalk was particularly popular among educators in the 1980s and early

'90s for classroom multimedia presentations. Although Hypertalk had many features of object-oriented languages (described in the next section), Apple did not develop it for other computer platforms and let it languish; as Apple's market share declined in the 1990s, a new cross-platform way of displaying multimedia left Hypertalk all but obsolete.

Object-oriented languages

Object-oriented languages help to manage complexity in large programs. Objects package data and the operations on them so that only the operations are publicly accessible and internal details of the data structures are hidden. This information hiding made large-scale programming easier by allowing a programmer to think about each part of the program in isolation. In addition, objects may be derived from more general ones, "inheriting" their capabilities. Such an object hierarchy made it possible to define specialized objects without repeating all that is in the more general ones.

Object-oriented programming began with the Simula language (1967), which added information hiding to ALGOL. Another influential object-oriented language was Smalltalk (1980), in which a program was a set of objects that interacted by sending messages to one another.

C++

The C++ language, developed by Bjarne Stroustrup at AT&T in the mid-1980s, extended C by adding objects to it while preserving the efficiency of C programs. It has been one of the most important languages for both education and industrial programming. Large parts of many operating systems, such as the Microsoft Corporation's Windows 98, were written in C++.

ADA

Ada was named for Augusta Ada King, countess of Lovelace, who was an assistant to the 19th-century English inventor Charles Babbage, and is sometimes called the first computer programmer. Ada, the language, was developed in the early 1980s for the U.S. Department of Defense for large-scale programming. It combined Pascal-like notation with the ability to package operations and data into independent modules. Its first form, Ada 83, was not fully object-oriented, but the subsequent Ada 95 provided objects and the ability to construct hierarchies of them. While no longer mandated for use in work for the Department of Defense, Ada remains an effective language for engineering large programs.

JAVA

In the early 1990s, Java was designed by Sun Microsystems, Inc., as a programming language for the World Wide Web (WWW). Although it resembled C++ in appearance, it was fully object-oriented. In particular, Java dispensed with lower-level features, including the ability to manipulate data addresses, a capability that is neither desirable nor useful in programs for distributed systems. In order to be portable, Java programs are translated by a Java Virtual Machine specific to each computer platform, which then executes the Java program. In addition to adding interactive capabilities to the Internet through Web "applets," Java has been widely used for programming small and portable devices, such as mobile telephones.

Visual Basic

Visual Basic was developed by Microsoft to extend the capabilities of BASIC by adding objects and "event-driven" programming: buttons, menus, and other elements of graphical user interfaces (GUIs). Visual Basic can also be used within other Microsoft software to program small routines.

Declarative languages

Declarative languages, also called nonprocedural or very high level, are programming languages in which (ideally) a program specifies what is to be done rather than how to do it. In such languages there is less difference between the specification of a program and its implementation than in the procedural languages described so far. The two common kinds of declarative languages are logic and functional languages.

Logic programming languages, of which PROLOG (programming in logic) is the best known, state a program as a set of logical relations (e.g., a grandparent is the parent of a parent of someone). Such languages are similar to the SQL database language. A program is executed by an "inference engine" that answers a query by searching these relations systematically to make inferences that will answer a query. PROLOG has been used extensively in natural language processing and other AI programs.

Functional languages have a mathematical style. A functional program is constructed by applying functions to arguments. Functional languages, such as LISP, ML, and Haskell, are used as research tools in language development, in automated mathematical theorem provers, and in some commercial projects.

Scripting languages

Scripting languages are sometimes called little languages. They are intended to solve relatively small programming problems that do not require the overhead of data declarations and other features needed to make large programs manageable. Scripting languages are used for writing operating system utilities, for special-purpose file-manipulation programs, and, because they are easy to learn, sometimes for considerably larger programs.

PERL (practical extraction and report language) was developed in the late 1980s, originally for use with the UNIX operating system. It was intended to have all the capabilities of earlier scripting languages. PERL provided many ways to state common operations and thereby allowed a programmer to adopt any convenient style. In the 1990s it became popular as a system-programming tool, both for small utility programs and for prototypes of larger ones. Together with other languages discussed below, it also became popular for programming computer Web "servers."

Document formatting languages

Document formatting languages specify the organization of printed text and graphics. They fall into several classes: text formatting notation that can serve the same functions as a word processing program, page description languages that are interpreted by a printing device, and, most generally, markup languages that describe the intended function of portions of a document.

TEX

TeX was developed during 1977–86 as a text formatting language by Donald Knuth, a Stanford University professor, to improve the quality of mathematical notation in his books. Text formatting systems, unlike WYSIWYG ("What You See Is What You Get") word processors, embed plain text formatting commands in a document, which are then interpreted by the language processor to produce a formatted document for display or printing. TeX marks italic text, for example, as {\it this is italicized}, which is then displayed as *this is italicized.*

TeX largely replaced earlier text formatting languages. Its powerful and flexible abilities gave an expert precise control over such things as the choice of fonts, layout of tables, mathematical notation, and the inclusion of graphics within a document. It is generally used with the aid of "macro" packages that define simple commands for common operations, such as starting a new paragraph; LaTeX is a widely used package. TeX contains numerous standards "style sheets" for different types of documents, and these may be further adapted by each user. There are also related programs such as BibTeX, which manages bibliographies and has style sheets for all of the common bibliography styles, and versions of TeX for languages with various alphabets.

POSTSCRIPT

PostScript is a page-description language developed in the early 1980s by Adobe Systems Incorporated on the basis of work at Xerox PARC (Palo Alto Research Center). Such languages describe documents in terms that can be interpreted by a personal computer to display the document on its screen or by a microprocessor in a printer or a typesetting device.

PostScript commands can, for example, precisely position text, in various fonts and sizes, draw images that are mathematically described, and specify colour or shading. PostScript uses postfix, also called reverse Polish notation, in which an operation name follows its arguments. Thus, "300 600 20 270 arc stroke" means: draw ("stroke") a 270-degree arc with radius 20 at location (300, 600). Although PostScript can be read and written by a programmer, it is normally produced by text formatting programs, word processors, or graphic display tools.

The success of PostScript is due to its specification's being in the public domain and to its being a good match for high-resolution laser printers. It has influenced the development of printing fonts, and manufacturers produce a large variety of PostScript fonts.

SGML

SGML (standard generalized markup language) is an international standard for the definition of markup languages; that is, it is a Meta language. Markup consists of notations called tags that specify the function of a piece of text or how it is to be displayed. SGML emphasizes descriptive markup, in which a tag might be "<emphasis>." Such a markup denotes the document function, and it could be interpreted as reverse video on a computer screen, underlining by a typewriter, or italics in typeset text.

SGML is used to specify DTDs (document type definitions). A DTD defines a kind of document, such as a report, by specifying what elements must appear in the document—e.g.

World Wide Web display languages

HTML

The World Wide Web is a system for displaying text, graphics, and audio retrieved over the Internet on a computer monitor. Each retrieval unit is known as a Web page, and such pages frequently contain "links" that allow related pages to be retrieved. HTML (*h*yper*t*ext *m*arkup *l*anguage) is the markup language for encoding Web pages. It was designed by Tim Berners-Lee at the CERN nuclear physics laboratory in Switzerland during the 1980s and is defined by an SGML DTD. HTML markup tags specify document elements such as headings, paragraphs, and tables. They mark up a document for display by a computer program known as a Web browser. The browser interprets the tags, displaying the headings, paragraphs, and tables in a layout that is adapted to the screen size and fonts available to it.

HTML documents also contain anchors, which are tags that specify links to other Web pages. An anchor has the form Encyclopedia Britannica, where the quoted string is the URL (universal resource locator) to which the link points (the Web "address") and the text following it is what appears in a Web browser, underlined to show that it is a link to another page. What is displayed as a single page may also be formed from multiple URLs, some containing text and others graphics.

XML

HTML does not allow one to define new text elements; that is, it is not extensible. XML (extensible markup language) is a simplified form of SGML intended for documents that are published on the Web. Like SGML, XML uses DTDs to define document types and the meanings of tags used in them. XML adopts conventions that make it easy to parse, such as that document entities are marked by both a beginning and an ending tag, such as …. XML provides more kinds of hypertext links than HTML, such as bidirectional links and links relative to a document subsection.

Because an author may define new tags, an XML DTD must also contain rules that instruct a Web browser how to interpret them—how an entity is to be displayed or how it is to generate an action such as preparing an e-mail message.

WEB SCRIPTING

Web pages marked up with HTML or XML are largely static documents. Web scripting can add information to a page as a reader uses it or let the reader enter information that may, for example, be passed on to the order department of an online business. CGI (common gateway interface) provides one mechanism; it transmits requests and responses between the reader's Web browser and the Web server that provides the page. The CGI component on the server contains small programs called scripts that take information from the browser system or provide it for display. A simple script might ask the reader's name, determine the Internet address of the system that the reader uses, and print a greeting. Scripts may be written in any programming language, but, because they are generally simple text-processing routines, scripting languages like PERL are particularly appropriate.

Another approach is to use a language designed for Web scripts to be executed by the browser. JavaScript is one such language, designed by the Netscape Communications Corp., which may be used with both Netscape's and Microsoft's browsers. JavaScript is a simple language, quite different from Java. A JavaScript program may be embedded in a Web page with the HTML tag <script language="JavaScript">. JavaScript instructions following that tag will be executed by the browser when the page is selected. In order to speed up display of dynamic (interactive) pages, JavaScript is often combined with XML or some other language for exchanging information between the server and the client's browser. In particular, the XMLHttpRequest command enables asynchronous data requests from the server without requiring the server to resend the entire Web page. This approach, or "philosophy," of programming is called Ajax (*a*synchronous *J*avaScript *a*nd *X*ML).

VB Script is a subset of Visual Basic. Originally developed for Microsoft's Office suite of programs, it was later used for Web scripting as well. Its capabilities are similar to those of JavaScript, and it may be embedded in HTML in the same fashion.

Behind the use of such scripting languages for Web programming lies the idea of component programming, in which programs are constructed by combining independent previously written components without any further language processing. JavaScript and VB Script programs were designed as components that may be attached to Web browsers to control how they display information.

Check your progress

Q1. What is the need of machine language?

Q2. Define business oriented languages.

1.4 Summary

In this unit you learnt about basics of Software, Types of Software, and Programming Languages.

- Software is any set of machine-readable instructions that directs a computer's processor to perform specific operations.
- System software is a program that manages and supports the computer resources and operations of a computer system
- Application software consists of Programs that direct computers to perform specific information processing activities for end users.
- Word processing software is used to create, manipulate, and print documents.
- A spreadsheet is an electronic worksheet used to organize and manipulate numbers and display options for what-if analysis.

1.5 Review Questions

Q1. What do you mean by software? How many types of software available?

Q2. What is the difference between system software and application software?

Q3. Define programming language with example.

Q4. What is object-oriented language?

Q5. Differentiate between object oriented and object based language.

UNIT-II

Software Qualities

Structure

2.0 Introduction

2.1 Software Nature & Qualities

2.2 Summary

2.3 Review Questions

2.0 Introduction

In this unit we focused on software qualities. There are three sections in this unit. In the first sections i.e. Sec. 2.1 you will learn about software nature and its qualities. The goal of any engineering activity is to build something—a product. The civil engineer builds a bridge, the aerospace engineer builds an airplane, and the electrical engineer builds a circuit. The product of software engineering is a "software system." It is not as tangible as the other products, but it is a product nonetheless. It serves a function. In some ways software products are similar to other engineering products, and in some ways they are very different. The characteristic that perhaps sets software apart from other engineering products the most is that software is malleable. We can modify the product itself—as opposed to its design—rather easily. This makes software quite different from other products such as cars or ovens. The malleability of software is often misused. While it is certainly possible to modify a bridge or an airplane to satisfy some new need—for example, to make the bridge support more traffic or the airplane carry more cargo—such a modification is not taken lightly and certainly is not attempted without first making a design change and verifying the impact of the change extensively. In Sec. 2.2 and 2.3 you will find summary and review questions respectively.

Objectives

After studying this unit you should be able to:

- Express software nature.
- Define software qualities.

2.1 Software Nature & Qualities

In the recent past, when bank statements contained errors or the telephone network broke down, the general public usually blamed "the computer," making no distinction between hardware and software. However, high-profile disasters and the ensuing debates in the press are alerting more people to the crucial nature of software quality in their everyday lives. Before long, we can expect increasing public concern about the pervasiveness of software, not only in public services but also in consumer products like automobiles, washing machines, telephones, and electric shavers. Consequently, we software professionals need to worry about the quality of all our products from large, complex, stand-alone systems to small embedded ones.

So how do we assess "adequate" quality in a software product? The context is important. Errors tolerated in word-processing software may not be acceptable in control software for a nuclear-power plant. Thus, we must reexamine the meanings of "safety-critical" and "mission-critical" in the context of software's contribution to the larger functionality and quality of products and businesses. At the same time, we must ask ourselves who is responsible for setting quality goals and making sure they are achieved.

WHAT DOES QUALITY REALLY MEAN?

Most of us are affected by the quality of the software we create because our organization's viability depends on it. And most software-related tools and methods including those described in IEEE Software claim to assess or improve software quality in some way. So we must question what we and our customers mean by software quality.

A good definition must let us measure quality in a meaningful way. Measurements let us know if our techniques really improve our software, as well as how process quality affects product quality. We also need to know how the quality we build in can affect the product's use after delivery and if the investment of time and resources to assure high quality reap higher profits or larger market share. In other words, we want to know if good software is good business.

Recent articles have raised this question, but the answer is still far from clear. Still, most people believe that quality is important and that it can be improved. Companies and countries continue to invest a great deal of time, money, and effort in improving software quality. But we should try to determine if these national initiatives have directly affected and improved software quality. The answer may depend on how you approach quality improvement. Some companies take a product-based approach, while others focus on process; both strategies have led to Malcolm Baldridge awards for overall product quality.

In their more general questioning of quality goals and techniques, Roger Howe, Dee Gaeddert, and Maynard Howe pointed out that most quality initiatives either fail (by drowning in a sea of rhetoric) or cannot demonstrate success because no financial return can be identified.1 In this special issue, we question software quality in the same way. We consider the meaning of software quality, how we assess it, and whether the steps we are taking to improve it are really worthwhile.

VIEWS OF SOFTWARE QUALITY

In an influential paper examining views of quality, David Garvin studied how quality is perceived in various domains, including philosophy, economics, marketing, and operations management.2 He concluded that "quality is a complex and multifaceted concept" that can be described from five different perspectives. The transcendental view sees quality as something that can be recognized but not defined.

The user view sees quality as fitness for purpose. The manufacturing view sees quality as conformance to specification. The product view sees quality as tied to inherent characteristics of the product. The value-based view sees quality as dependent on the amount a customer is willing to pay for it.

Transcendental view

This view of software quality is much like Plato's description of the ideal or Aristotle's concept of form. Just as every table is different but each is an approximation of an ideal table, we can think of software quality as something toward which we strive as an ideal, but may never implement completely. When software gurus exhort us to produce products that delight users, this delight represents the strived-for "recognition" in the transcendental definition of quality.

User view

Whereas the transcendental view is ethereal, the user view is more concrete, grounded in product characteristics that meet the user's needs. This view of quality evaluates the product in a task context and can thus be a highly personalized view. In reliability and performance modeling, the user view is inherent, since both methods assess product behavior with respect to operational profiles (that is, to expected functionality and usage patterns). Product usability is also related to the user view: in usability laboratories, researchers observe how users interact with software products.

Manufacturing view

The Manufacturing view focuses on product quality during production and after delivery. This view examines whether or not the product was constructed "right the first time," in an effort to avoid the costs associated with rework during development and after delivery. This process focus can lead to quality assessment that is virtually independent of the product itself. That is, the manufacturing approach adopted by ISO 9001 and the Capability Maturity Model advocates conformance to process rather than to specification.

There is little evidence that conformance to process standards guarantees good products. In fact, critics of this view suggest that process standards guarantee only uniformity of output and can thus institutionalize the production of mediocre or bad products. However, this criticism may be unfair. Although process standards are usually based on the principle of "documenting what you do and doing what you say," both CMM and ISO 9001 also insist (with different degrees of emphasis) that you improve your process to enhance product quality.

Product view

Whereas the user and manufacturing views examine the product from without, a product view of quality looks inside, considering the product's inherent characteristics. This approach is frequently adopted by software-metrics advocates, who assume that measuring and controlling internal product properties (internal quality indicators) will result in improved external product behavior (quality in use). Assessing quality by measuring internal properties is attractive because it offers an objective and context independent view of quality. However, more research is needed to confirm that internal quality assures external quality and to determine which aspects of internal quality affect the product's use. Some researchers have developed models to link the product view to the user view.

Value-based view

Different views can be held by different groups involved in software development. Customers or marketing groups typically have a user view, researchers a product view, and the production department a manufacturing view. If the difference in viewpoints is not made explicit is understandings about quality created during project initiation are likely to resurface as (potentially) major problems during product acceptance.

These disparate views can complement each other in early phases. If the user's view is stated explicitly during requirements specification, the technical specification that drives the production process can be derived directly from it as can product functionality and features. However, problems can arise when changes to the requirements occur. At this point, the user's requirement for a useful product may be in conflict with the manufacturer's goal of minimizing rework.

This is where the value-based view of quality becomes important. Equating quality to what the customer is willing to pay for encourages everyone to consider the trade-offs between cost and quality. A value-based perception can involve techniques to manage conflicts when requirements change. Among them are "design to cost," in which design possibilities are constrained by available resources and "requirements scrubbing," in which requirements are assessed and revised in light of costs and benefits.

Product purchasers take a rather different value-based view. Part of their job is to know if a software product represents value for money to their organization. In this context, internal software measures are irrelevant. Purchasers compare the product cost with the potential benefits.

Measuring the user's view

When users think of software quality, they often think of reliability: how long the product functions properly between failures. Reliability models plot the number of failures over time. These models sometimes use an operational profile, which depicts the likely use of different system functions.

Users, however, often measure more than reliability. They are also concerned about usability, including ease of installation, learning, and use. Tom Gilb suggests that these characteristics can be measured directly. For example, learning time can be captured as the average elapsed time (in hours) for a typical user to achieve a stated level of competence. Gilb's technique can be generalized to any quality feature. The quality concept is broken down into component parts until each can be stated in terms of directly measurable attributes. Thus, each quality-requirement specification includes a measurement concept, unit, and tool, as well as the planned level (the target for good quality), the currently available level, the best possible level (state-of-the-art), and worst level. Gilb does not prescribe a universal set of quality concepts and Measurements, because different systems will require different qualities and different measurements.

Measuring the manufacturer's view

The manufacturing view of quality suggests two characteristics to measure: defect counts and rework costs.

Defect counts. Defect counts are the number of known defects recorded against a product during development and use. For comparison across modules, products, or projects, you must count defects in the same way and at the same time during the development and maintenance processes. For more detailed analysis, you can categorize defects on the basis of the phase or activity where the defect was introduced, as well as the phase or activity in which it was detected. This information can be especially helpful in evaluating the effects of process change (such as the introduction of inspections, tools, or languages). The relationship between defects counts and operational failures is unclear. However, you can use defect counts to indicate test efficiency and identify process-improvement areas. In addition, a stable environment can help you estimate post-release defect counts.

To compare the quality of different products, you can "normalize" defect count by product size, to yield a defect density. This measure lets you better compare modules or products that differ greatly in size. In addition, you can "normalize" post-release defect counts by the number of product users, the number of installations, or the amount of use. Dividing the number of defects found during a particular development stage by the total number of defects found during the product's life helps determine the effectiveness of different testing activities.

Rework costs. Defects differ in their effect on the system: some take a little time to find and fix; others are catastrophic and consume valuable resources. To monitor the effect of defect detection and correction, we often measure rework costs the staff effort spent correcting defects before and after release. This cost of nonconformance supports the manufacturing view. Rework is defined as any additional effort required to find and fix problems after documents and code are formally signed-off as part of configuration management. Thus, end-phase verification and validation are usually excluded, but debugging effort during integration and system testing is included. To compare different products, rework effort is sometimes "normalized" by being calculated as a percentage of development effort. Because we want to capture the cost of nonconformance, we must be sure to distinguish effort spent on enhancements from effort spent on maintenance. Only defect correction should count as rework. It is also important to separate pre- and post-release rework. Post release rework effort is a measure of delivered quality; prerelease rework effort is a measure of manufacturing efficiency. If we can attribute the prerelease rework effort to specific phases, we can use it to identify areas for process improvement.

Developers and customers alike are interested in knowing as early as possible the likely quality of the delivered product. But the relationship between postdelivery failure and defects, structural measures, and other predelivery information is far from clear. In 1984, the Esprit-funded Request project concluded that there were no software-product metrics that were likely to be good predictors of final product qualities. Twelve years later, there is no evidence of any significant improvement. Much useful software-metrics research concentrates instead on linking software product measures to error-prone modules.

Capturing quality data

The way we measure quality depends on the viewpoint we take and the aspect of quality we want to capture. Peter Mellor provides guidelines for defining incidents, failures, and faults that can help you capture raw data for reliability assessment. This type of data can measure other aspects related to the user view of quality. Proper classification of incidents lets us identify potential usability problems (that is, incidents resulting from misuse of the software or misunderstanding of the user manuals and help systems). In addition, information about the time and effort needed to diagnose the cause of different priorities and correct any underlying faults can give us useful information about system maintainability. This sort of data is often used to monitor service-level agreements that define the obligations of software-maintenance organizations.

Capturing data associated with other quality aspects particularly those associated with the product and manufacturing view is usually part of a company's software measurement system. The particular measures an organization collects will depend on its goals and management requirements. Techniques such as the Goal-Question-Metric paradigm developed by Vic Basili and colleagues can help us identify which measures will help us monitor and improve quality.

THE BUSINESS VALUE OF QUALITY

In the last few decades, software has grown to become a vital part of most companies' products and services. With that growth comes our responsibility for determining how much software contributes to the corporate bottom line. When a telephone company cannot implement a new service because the billing-system software cannot handle the new features, then lack of software quality is a corporate problem. When a national gas utility must spend millions of dollars to fix a software glitch in monitoring systems embedded in gas meters throughout the country, then small software defects become big headaches. And when software problems stop the assembly line, ground the plane, or send the troops to the wrong location, organizations realize that software is essential to the health and safety of business and people. Little research is done into the relationship between software quality and business effectiveness and efficiency. But unless we begin to look at these issues, companies will be unable to support key business decisions.

In particular, we must look more carefully at how our methods and tools affect software quality. Businesses take big risks when they invest in technology that has not been carefully tested and evaluated. The Desmet project, funded by the UK's Department of Trade and Industry, has produced guidelines for how to conduct case studies and experiments in support of technology evaluation. But looking at the software alone is not enough. We must see it in the context of how it is used by business to determine if investment in higher software quality is worthwhile. As Ed Yourdon pointed out, sometimes less-than-perfect is good enough; only business goals and priorities can determine how much "less than perfect" we are willing to accept. In their article, "Software Quality in

Consumer Electronic Products," Jan Rooijmans and colleagues take up this issue with a discussion of the problems and challenges associated with producing consumer electronic products.

Check your progress

Q1. What is the natures of software?

Q2. Define the qualities of software.

2.2 Summary

In this unit you learnt about software natures and its qualities. Software paly a very important role in computer science and of courses, software should be perfect in terms of nature and quality.

- Software professionals need to worry about the quality of all our products from large, complex, stand-alone systems to small embedded ones.
- A good definition must let us measure quality in a meaningful way.
- The manufacturing view sees quality as conformance to specification.
- Transcendental view of software quality is much like Plato's description of the ideal or Aristotle's concept of form.
- Whereas the transcendental view is ethereal, the user view is more concrete, grounded in product characteristics that meet the user's needs.

2.3 Review Questions

Q1. Define the nature of software.

Q2. What are the qualities of good software?

Q3. How can we develop software in term of good business quality?

Q4. Define the design quality of the software.

UNIT-III

Operating System

Structure

3.0 Introduction

3.1 Operating Systems

3.2 History and Evolution

3.3 Main functions of Operating System

3.4 Multitasking

3.5 Summary

3.6 Review Questions

3.0 Introduction

This unit focused on the Operating system. In this unit there are six sections. In the first section i.e. Sec. 3.1 we defined operating System. What is an operating system? An operating system (sometimes abbreviated as "OS") is the program that, after being initially loaded into the computer by a boot program, manages all the other programs in a computer. The other programs are called applications or application programs. The application programs make use of the operating system by making requests for services through a defined application program interface (API). In addition, users can interact directly with the operating system through a user interface such as a command language or a graphical user interface (GUI). In the Sec. 3.2 you will know about history & evaluations. In Sec. 3.3 you will learn about different functions like Operating System as a Resource Manager, process management, storage management, memory management etc. In Sec. 3.4 we defined multitasking. In multitasking, only one CPU is involved, but it switches from one program to another so quickly that it gives the appearance of executing all of the programs at the same time. In Sec. 3.5 and 3.6 you will find summary and review questions respectively.

Objectives

After studying this unit you should be able to:

- Define Operating Systems.
- Describe History and Evolution.
- Express Main functions of OS and Multitasking.

3.1 Operating System

An operating system is a program that runs on a computer to simplify the use of the computer for the user. The operating system manages the use of peripheral devices such as printers, monitors and keyboards. In addition the operating system will run other programs and display the results. In order to carry out these functions the operating system has to require a systematic structure for the inputs and outputs; there is a definite structure to files and there is a systematic way in which the files are stored on the data storage devices. Without an operating system a computer is largely an unresponsive hunk of metal and wires.

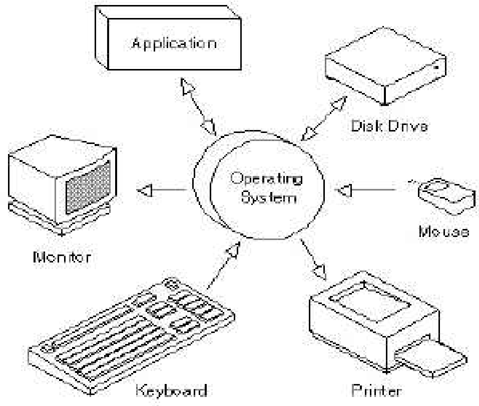


Figure 3.1: - Operating System

3.2 History and Evaluation of Operating System

Operating Systems are at the core of any modern technological advancement. Operating Systems set the rules of engagement for other programmers by defining what can be done and the PC's limitations. They are a creation of programmers, for programmers. They enable other programmers to do their job easier, as they do all the low-level operations such as interfacing with the hardware.

If you think of programs as stacked on top of each other, just like a house of cards, Operating Systems are the bottom and most important layer. An OS defines how broad and capable all the other programs will be. If an OS is limiting due to bugs or programmer's decisions (such as the Unix "Y2K38" bug), then so will the applications stacked on top of it.

Here are notable Operating Systems in our evolution of computers.

1956, GM-NAA I/O: Developed by Robert L. Patrick of General Motors for use on their IBM 704 mainframe. This early OS was primarily designed to automatically switch to the next job once its current job was completed. It was used on about forty IBM 704 mainframes.

1961, MCP (Master Control Program): Developed by Burroughs Corporations for their B5000 mainframe. MCP is still in used today by the Unisys ClearPath/MCP machines.

1966, DOS/360: After years of being strictly in the hardware business, IBM ventured into the OS. IBM developed a few unsuccessful mainframe Operating Systems until it finally released DOS/360 and its successors, which put IBM in the driver seat for both the hardware and OS industries.

1969, Unix: Developed by AT&T Bell Labs programmers Ken Thompson, Dennis Ritchie, Douglas McIlroy, and Joe Ossanna. It gained widespread acceptance first within the large AT&T company, and later by colleges and universities. It is written in C, which allows for easier modification, acceptance, and portability.

1973, CP/M (Control Program/Monitor (later re-purposed as "Control Program for Microcomputers"): Developed by Greg Kildall as a side project for his company Digital Research. CP/M became a popular OS in the 1970's. It had many applications developed for it, including WordStar and dBASE. It was ported to a variety of hardware environments. In fact, IBM originally wanted CP/M for its new Personal Computers, but later selected MS-DOS when a deal could not be reached.

1977, BSD (Berkeley Software Distribution): Developed by the University of California, Berkeley. BSD is a Unix variant based on early versions of Unix from Bell Labs.

1981, MS-DOS: Developed by Microsoft for the IBM PC's. It was the first widely available Operating Systems for home users. In 1985, Microsoft released Microsoft Windows, which popularized the Operating System even more. Microsoft Windows allowed users a graphical user interface (GUI), which rapidly spread Microsoft's product.

1982, SunOS: Developed by Sun Microsystems, SunOS was based on BSD. It was a very popular Unix variant.

1984, Mac OS: Developed by Apple Computer, Inc for their new product, the Macintosh home PC. The Macintosh was widely advertised (the famous 1984 commercial is available below). Mac OS was the first OS with a GUI built-in. This lead to a very stable OS, as well as wide acceptance due to its ease of use

1987, OS/2: Developed by a joint venture of IBM and Microsoft. Though the OS was heavily marketed, it did not pick up in popularity.

1991, Linux: Developed by Linus Torvalds as a free UNIX variant. Linux today is a very largely contributed Open Source project that plays a very prominent role in today's server industry.

1992, Sun Solaris: Developed by Sun Microsystems, Solaris is a widely used Unix variant, and partially developed based on Sun's SunOS.

1993, Windows NT: Developed by Microsoft as a high-end server Operating System, the NT code became the basis for Operating Systems to this day. NT was primarily used on computers used as servers to counter the Unix dominance in the arena.

1995, Windows95: Developed by Microsoft, it was the first Microsoft Operating system to have a graphical user interface built into it. It was tremendously marketed (successfully) and quickly swept across the country and the globe. Below is one of Microsoft's popular commercials, featuring the Rolling Stones with "Start Me Up", drawing attention to Microsoft's "Start" button, which to this day is a dominant feature of their Operating Systems.

1997, JavaOS: Developed by Sun Microsystems, JavaOS was developed primarily using the Java programming language. The OS was created to be installed on any device, including PC's.

1998, Windows98: Developed by Microsoft, Windows 98 was the next iteration of the Microsoft Windows95 Operating System.

1999, MacOS X Server 1.0: Developed by Apple Computer, Inc., MacOS X Server 1.0 was a precursor to Apple's MacOS X desktop version, which replaced it in 2001. MacOS X Server 1.0 was developed for Apple's popular Macintosh PC.

2000, Windows 2000: Developed by Microsoft, Windows 2000 was a much improved Operating System over Windows 98. It was developed from a dramatically different code base. It was targetted for business oriented uses.

2000, Windows ME: Developed by Microsoft, Windows ME (also called Windows Millenium) was a rather unsuccessful new version of Windows 98 and had a short shelf life. It was released just seven months after Windows 2000 and just a year before Windows XP.

2001, MacOS X Version 10.0: Developed by Apple Computer, Inc., MacOS X Version 10.0 dramatically changed the user interface for Apple's Macinstosh users.

2001, Windows XP: Developed by Microsoft, Windows XP was an enhanced version of Windows 2000 code base. XP became widely popular and is used extensively today, despite the release of newer versions of Windows.

2003, Windows Server 2003: Developed by Microsoft as an improved version of their NT OS.

2007, Windows Vista: Developed by Microsoft, Windows Vista had been slow in taking off.

2008, Windows Server 2008: Developed by Microsoft as an upgrade to Windows Server 2003.

2009, Windows 7: Developed by Microsoft to replace Vista, "Win7" is currently used by over 50% of internet users.

2012, Windows 8: Developed by Microsoft to replace Win7, "Win8" was just released October 26th, 2012.

EVOLUTION OF OPERATING SYSTEMS

Serial Processing

Users access the computer in series. From the late 1940's to mid-1950's, the programmer interacted directly with computer hardware i.e., no operating system. These machines were run with a console consisting of display lights, toggle switches, some form of input device and a printer. Programs in machine code are loaded with the input device like card reader. If an error occurs the program was halted and the error condition was indicated by lights. Programmers examine the registers and main memory to determine error. If the program is success, then output will appear on the printer.

Main problem here is the setup time. That is single program needs to load source program into memory, saving the compiled (object) program and then loading and linking together.

Simple Batch Systems

To speed up processing, jobs with similar needs are batched together and run as a group. Thus, the programmers will leave their programs with the operator. The operator will sort programs into batches with similar requirements.

The problems with Batch Systems are:

- Lack of interaction between the user and job.
- CPU is often idle, because the speeds of the mechanical I/O devices are slower than CPU.

For overcoming this problem use the Spooling Technique. Spool is a buffer that holds output for a device, such as printer, that cannot accept interleaved data streams. That is when the job requests the printer to output a line that line is copied into a system buffer and is written to the disk. When the job is completed, the output is printed. Spooling technique can keep both the CPU and the I/O devices working at much higher rates.

Multiprogrammed Batch Systems

More than one program resides in the main memory. While a program A uses an I/O device the processor does not stay idle, instead it runs another program B.

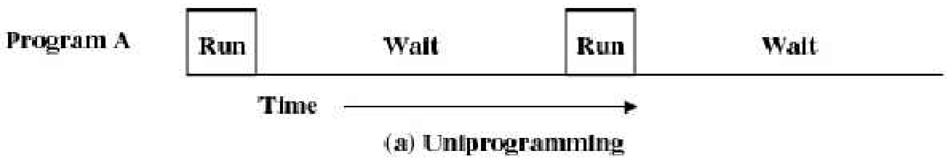


Figure 3.2: - Uni-programming

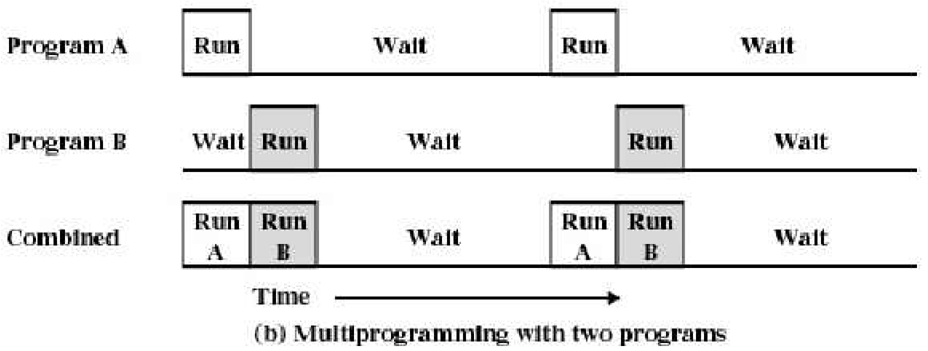


Figure 3.3: - Multiprogramming

New features:

Memory management - to have several jobs ready to run, they must be kept in main memory

Job scheduling - the processor must decide which program to run.

Time-Sharing Systems

Time-sharing systems are not available in 1960s. Time-sharing or multitasking is a logical extension of multiprogramming. That is processors time is shared among multiple users simultaneously is called time-sharing. The main difference between Multiprogrammed Batch Systems and Time-Sharing Systems is in multiprogrammed batch systems its objective is maximize processor use, whereas in Time-Sharing Systems its objective is minimize response time.

Multiple jobs are executed by the CPU by switching between them, but the switches occur so frequently. Thus, the user can receive an immediate response. For example, in a transaction processing, processor execute each user program in a short burst or quantum of computation. That is if n users are present, each user can get time quantum. When the user submits the command, the response time is seconds at most.

Operating system uses CPU scheduling and multiprogramming to provide each user with a small portion of a time. Computer systems that were designed primarily as batch systems have been modified to time-sharing systems.

For example IBM's OS/360

Time-sharing operating systems are even more complex than multiprogrammed operating systems. As in multiprogramming, several jobs must be kept simultaneously in memory.

Personal-Computer Systems (PCs)

A computer system is dedicated to a single user is called personal computer, appeared in the 1970s. Micro computers are considerably smaller and less expensive than mainframe computers. The goals of the operating system have changed with time; instead of maximizing CPU and peripheral utilization, the systems developed for maximizing user convenience and responsiveness.

For e.g., MS-DOS, Microsoft Windows and Apple Macintosh

Hardware costs for microcomputers are sufficiently low. Decrease the cost of computer hardware (such as processors and other devices) will increase our needs to understand the concepts of operating system. Malicious programs destroy data on systems. These programs may be self-replicating and may spread rapidly via worm or virus mechanisms to disrupt entire companies or even worldwide networks.

MULTICS operating system was developed from 1965 to 1970 at the Massachusetts Institute of Technology (MIT) as a computing utility. Many of the ideas in MULTICS were subsequently used at Bell Laboratories in the design of UNIX OS.

Parallel Systems

Most systems to date are single-processor systems; that is they have only one main CPU. Multiprocessor systems have more than one processor.

The advantages of parallel system are as follows:

- throughput (Number of jobs to finish in a time period)
- Save money by sharing peripherals, cabinets and power supplies
- Increase reliability
- Fault-tolerant (Failure of one processor will not halt the system).

Symmetric multiprocessing model

Each processor runs an identical job (copy) of the operating system, and these copies communicate. Encore's version of UNIX operating system is a symmetric model
e.g. If two processors are connected by a bus. One is primary and the other is the backup. At fixed check points in the execution of the system, the state information of each job 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. But it is expensive.

Asymmetric multiprocessing model

Each processor is assigned a specific task. A master processor controls the system. Sun's operating system SunOS version 4 is a asymmetric model. Personal computers contain a microprocessor in the keyboard to convert the key strokes into codes to be sent to the CPU.

Distributed Systems

Distributed systems distribute computation among several processors. In contrast to tightly couple systems (i.e., parallel systems), the processors 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 are referred as loosely coupled systems or distributed systems. Processors in a distributed system may vary in size and function. These processors are referred as sites, nodes, and computers and so on.

The advantages of distributed systems are as follows:

- Resource Sharing: With resource sharing facility user at one site may be able to use the resources available at another.

- Communication Speedup: Speedup the exchange of data with one another via electronic mail.
- Reliability: If one site fails in a distributed system, the remaining sites can potentially continue operating.

Real-time Systems

Real-time systems are used when there are rigid time requirements on the operation of a processor or the flow of data and real-time systems can be used as a control device in a dedicated application. Real-time operating system has well-defined, fixed time constraints otherwise system will fail.

For example Scientific experiments, medical imaging systems, industrial control systems, weapon systems, robots, and home-appliance controllers.

There are two types of real-time systems:

- **Hard real-time systems**

A hard real-time system guarantees that critical tasks complete on time. In hard real-time systems secondary storage is limited or missing with data stored in ROM. In these systems virtual memory is almost never found.

- **Soft real-time systems**

Soft real time systems are less restrictive. Critical real-time task gets priority over other tasks and retains the priority until it completes. Soft real-time systems have limited utility than hard real-time systems.

For example Multimedia, virtual reality, Advanced Scientific Projects like undersea exploration and planetary rovers.

3.3 Functions of Operating System

The main functions performed by most OS are as follows:

Process Management

Processing jobs deciding on the job scheduling technique and how long a job is to be processed, releasing the processor when the jobs are terminated.

Memory Management

As a memory manager, the OS handles the allocation and deallocation of memory space as required by various programs.

Device Management

OS provides I/O subsystem between process and device driver. It also detects device failures and notifies the same to the user.

File Management

OS is responsible to creation, deletion of files and directories. It also takes care of other file related activates such as retrieving, naming, and protecting the files.

Security Management

OS protects system resources and information against destruction and from unauthorized use.

Command Interpretation

The command interpretation is the layer that actually interacts with the computer operator. It consists of set of commands through which user communicate the program.

In addition to the above listed major functions, an operating system also performs few other functions such as keeping an account of which users use what kind of computer resources and how much etc. The common functions of controlling and allocating resources are then brought together into one piece of software- is called the operating system.

Check your progress

Q1. Define Operating System. Also define its evaluation.

Q2. Explain different types of functions of Operating System.

3.4 Multitasking

Multitasking is single-user variation of multiprogramming concept. Many authors do not distinguish between Multiprogramming and multitasking because both refer to the same concept of a system's capability to work concurrently on more than one task (job or process). Whenever a task (job or process) needs to perform I/O operations, the system uses it's CPU for executing some other task (job or process) that is also residing in the stem and is ready to use the CPU. However, some authors prefer to use the term multiprogramming for multi-user systems (multiple users can use such a system simultaneously, such as a mainframe system or a server class system), and multitasking for single-user systems (only one user uses such a system at a time, such as a personal computer or a notebook computer). Note that even in a single-user system, the system often processes multiple tasks at a time. For example, a personal computer user can execute a sorting job in background, while editing file in foreground. Similarly, a personal computer user may be reading his/her electronic mail in foreground; while compilation of a program is in progress in background. In this manner, a user may work concurrently on ma tasks. In a multitasking system, the user can partition the computer's display screen into multiple windows and can view the status of different tasks on different windows.

Multitasking eases user operation and saves lots of time when a user has to switch between two or more applications while performing a job. For example, let us assume that a user is using a word processor to create annual report and he/she needs to do some arithmetic calculations for including calculated results in the report. Without multitasking, the user would have to close the annual report file and word processing application, open calculator application, make necessary calculations, write down the results, close calculator application, reopen word processing application with annual report file, and embed calculation results in it. With multitasking, a user simply opens calculator application while working on annual report file creation, makes necessary calculations, and switches back to the annual report file to continue working on it.

Hence, for those who like to differentiate between multiprogramming and multitasking, multiprogramming interleaved execution of multiple jobs (of same or different users) in a multi-user system, while multitasking interleaved execution of multiple jobs (often referred to as tasks of same user) in a single-user system.

Check your progress

Q1. Explain multitasking. Why a system need multitasking?

3.5 Summary

In this unit you learnt about operating system, history and evaluation of operating system, operating system functions and multitasking.

- The operating system manages the use of peripheral devices such as printers, monitors and keyboards.
- Operating Systems set the rules of engagement for other programmers by defining what can be done and the PC's limitations.
- Programs in machine code are loaded with the input device like card reader.
- To speed up processing, jobs with similar needs are batched together and run as a group.
- Time-sharing or multitasking is a logical extension of multiprogramming.

3.6 Review Questions

Q1. What is an Operating system? Why we use an operating system?

Q2. "Operating System acts as a resource manager", justify your answer.

Q3. Write a short note on the evaluation of operating system.

Q4. What are the main functions of an operating system? Explain in detail.

Q5. What is multi-tasking operating system?

UNIT-IV

Operating System Types

Structure

4.0 Introduction

4.1 Multiprocessing

4.2 Time Sharing

4.3 Real Time Operating System

4.4 Summary

4.5 Review Questions

4.0 Introduction

In this unit we focused on types of Operating system. There are five sections in this unit. In the first sections i.e. Sec. 4.1 you will learn about multiprocessing. Multiprocessing is the use of two or more central processing units (CPUs) within a single computer system. The term also refers to the ability of a system to support more than one processor and/or the ability to allocate tasks between them. In Sec. 4.2 we defined time sharing. Time sharing is a technique which enables many people, located at various terminals, to use a particular computer system at the same time. Time-sharing or multitasking is a logical extension of multiprogramming. Processor's time which is shared among multiple users simultaneously is termed as time-sharing. The main difference between Multiprogrammed Batch Systems and Time-Sharing Systems is that in case of multi-programmed batch systems, objective is to maximize processor use, whereas in Time-Sharing Systems objective is to minimize response time. In Sec. 4.3 we introduced real time operating system. Real time system is defines as a data processing system in which the time interval required to process and respond to inputs is so small that it controls the environment. In Sec. 4.4 and 4.5 you will find summary and review questions respectively.

Objectives

After studying this unit you should be able to:

- Define Multiprocessing
- Describe Time Sharing
- Express Real Time Operating System

4.1 Multiprocessing

Up to this point, we have considered uniprocessor systems (having one CPU only). However, we have already seen that the use of I/O processors improves the efficiency of a computer system by making concurrent input, processing, and output operations possible. CPU performs arithmetic and logical operations, while I/O processors carry out I/O operations concurrently.

Designer carried further the idea of using I/O processors to improve system performance by designing systems with multiple CPUs. Such systems are called multiprocessing systems because they use multiple processors (CPUs) and can execute multiple processes concurrently. These systems use multiple CPUs to process either instructions from different and independent programs or different instructions from the same program simultaneously. Figure 4.1 shows basic organization of a typical multiprocessing system.

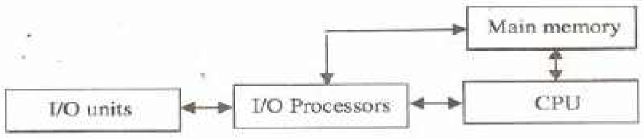


Figure Architecture of a computer system with its CPU, memory, and I/O processors.

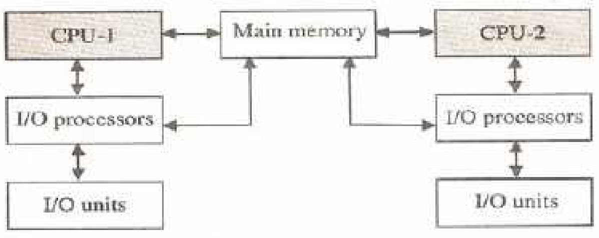


Figure Basic organization of a typical multiprocessing system.

Figure 4.1 Multi-process System

Multiprocessing systems are of two types — tightly-coupled systems and loosely-coupled systems. In tightly-coupled systems, all processors share a single system-wide primary memory. On the other hand, in loosely-coupled systems, the processors do not share memory, and each processor has its own local memory.

Difference between Multiprogramming and Multiprocessing

Multiprocessing is simultaneous execution of two or more processes by a computer system having more than one CPU. Comparatively, multiprogramming is interleaved execution of two or more processes by a single-CPU system. Hence, while multiprogramming involves execution of a portion of one program, then a portion of another, etc., in brief consecutive periods, multiprocessing involves simultaneous execution of several program segments of the same or different programs.

Advantages and Limitations of Multiprocessing

Multiprocessing systems have the following advantages:

1. **Better performance.** They have better performance (shorter response times and higher throughput) than single-processor systems. For example, if we have to run two different programs, a two-processor system is evidently more powerful than a single-processor system because the two-processor system can run both programs on different processors simultaneously. Similarly, if we can partition a large computation job into multiple sub-computations (sub-processes) that can run concurrently, a multiprocessor system with sufficient number of processors can run all sub-processes simultaneously, with each one on a

different processor (popularly known as parallel processing). However, the speed-up ratio with n processors is no n, but less than *n*. This is because when multiple processors cooperate to execute the processes or sub processes, the operating system incurs certain amount of overhead in keeping everything working correctly. This overhead, plus contention for shared resources, lowers the expected gains from additional processors.

2. **Better reliability.** They also have better reliability than single-processor systems. In a properly designed multiprocessor system, if one of the processors breaks down, the system can continue to function with remaining processors. Hence, the system can tolerate partial failure of processors and can avoid a complete breakdown. For example, if a system has 4 processors and one fails, then the system can utilize the remaining 3 processors to process the jobs. Thus, the entire system runs only 25% slower, rather than failing altogether. This ability of a system to continue providing service proportional to the level of non-failed hardware is called graceful degradation feature.

   Multiprocessing systems, however, require a very sophisticated operating system to schedule, balance, and coordinate the input, output, and processing activities of multiple processors. Designing such an operating system is complex and time taking. Moreover, multiprocessing systems have higher initial cost and their regular operation and maintenance is costlier than single-processor systems.

4.2 Time-sharing

Time-sharing is a mechanism to provide simultaneous interactive use of a computer system by many users in such a way that all users feel that he/she is the sole user of the system. It uses multiprogramming with a special CPU scheduling algorithm to achieve this.

A time-sharing system has many (even hundreds of) user terminals connected to it simultaneously. Using these terminals, multiple users can work on the system simultaneously. Multiprogramming feature allows multiple user programs to reside simultaneously in main memory, and special CPU scheduling algorithm allocates a short period of CPU time one-by-one to each user process (from the first one to the last one, and then again beginning from the first one). The short period during which a user process gets to use CPU is known as time slice, time slot, or quantum, and is typically of the order of 10 to 100 milliseconds. Hence, when the operating system allocates CPU to a user process, the process uses the CPU until its time slice expires (system's clock sends an interrupt signal to CPU after every time slice), or it needs to perform some I/O operation, or it completes its execution during this period. Notice that the operating system takes away CPU from a running process when its allotted time slice expires. Figure shows the process state diagram of a time-sharing system.

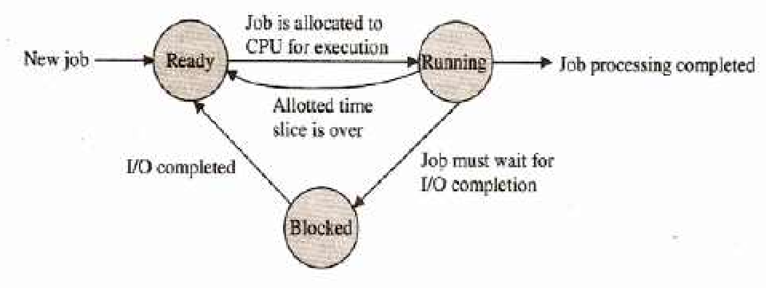

New job
Ready
Job is allocated to CPU for execution
Running
Job processing completed
Allotted time slice is over
I/O completed
Job must wait for I/O completion
Blocked

Figure Process state diagram for a time-sharing system.

Figure 4.2 Time Sharing System

Now let us see how the CPU scheduling algorithm, mentioned above, gives an impression to each user that he/she is the sole user of the system. Let the time slice be 10 milliseconds and the processing speed of the system's CPU be 500 million instructions per second. Hence, the system executes $500 \times 10^6 \times 10^{-3} \times 10 = 5 \times 10^6$ = 5 million instructions in 10 milliseconds. This is large enough for substantial progress of a single user process. Let there be 100 user terminals and 100 simultaneous users using the system. If the operating system allocates 10 milliseconds to each user process one-by-one, a user process will get CPU's attention once in every 100 x 10 milliseconds = I second. As human reaction time is of the order of a few seconds, a user will not notice any delay in execution of his/her commands and will normally feel that he/she is the sole user of the system, whereas in reality, many users are sharing a single computer.

Requirements of Time-sharing Systems

Time-sharing systems require following additional hardware and software:

1. A number of user terminals so that multiple users can use the system simultaneously in interactive mode.

2. Relatively large memory to support multiprogramming.

3. Memory protection mechanism to prevent a job's instructions and data from other jobs in a multiprogramming environment.

4. Job status preservation mechanism to preserve a job's status information when the operating system takes away CPU from it, and restores this information back, before it gives CPU to the process again.

5. A special CPU scheduling algorithm that allocates CPU for a short period one-by-one to each user process in a circular fashion.

6. An interrupt mechanism to send an interrupt signal to CPU after every time slice.

Advantages of Time-sharing Systems

Although, time-sharing systems are complex to design, they provide following advantages to their users:

1. **Reduce CPU idle time.** Thinking and typing speed of a user is much slower than processing speed of a computer. Hence, during interactive usage of a system, while a user is engaged in thinking or typing his/her input, a time-sharing system services many other users. Time-sharing systems, therefore, reduce CPU idle time and provide higher system throughput.

2. **Provide advantages of quick response time.** The special CPU scheduling algorithm used in time-sharing systems ensures quick response time to all users. This feature helps in improving programmer's efficiency by making interactive programming and debugging much simpler and quicker. Multiple programmers can work simultaneously for writing, testing, and debugging their programs, or for trying out various approaches to solve a problem. The greatest benefit of such a system is that all simultaneous users of the system can detect errors, correct them, and continue with their work immediately. This is in contrast to a batch system in which users have to correct errors offline and then resubmit their jobs for another run. We often measure in hours the time delay between job submission and return of the output in a batch system.

3. **Offer good computing facility to small users.** Small users can gain direct access to more sophisticated hardware and software than they could otherwise justify or afford. In

time-sharing systems, they merely pay a fee for the resources they use, and are relieved of hardware, software, and personnel problems, associated with acquiring and maintaining their own installation.

4.3 Real-Time Operating System

A **Real-Time Operating System** (RTOS) comprises of two components, viz., "Real-Time" and "Operating System".

An **Operating system** (OS) is nothing but a collection of system calls or functions which provides an interface between hardware and application programs. It manages the hardware resources of a computer and hosting applications that run on the computer. An OS typically provides multitasking, synchronization, Interrupt and Event Handling, Input/ Output, Inter-task Communication, Timers and Clocks and Memory Management. Core of the OS is the Kernel which is typically a small, highly optimized set of libraries.

Real-time systems are those systems in which the **correctness** of the system depends **not only** on the **logical result** of computation, **but also** on the **time** at which the results are produced.

RTOS is therefore an operating system that supports real-time applications by providing logically correct result within the deadline required. Basic Structure is similar to regular OS but, in addition, it provides mechanisms to allow real time scheduling of tasks.

Though real-time operating systems may or may not increase the speed of execution, they can provide much more precise and predictable timing characteristics than general-purpose OS.

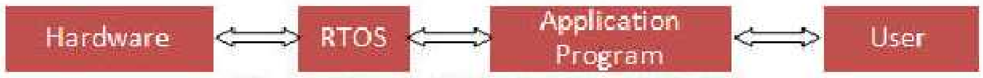


Fig. Real time embedded system with RTOS

Figure 4.3 : - Real-Time Operating System

RTOS is a key to many embedded systems and provides a platform to build applications. All embedded_systems are not designed with RTOS. Embedded systems with relatively simple/small hardware/code might not require an RTOS. Embedded systems with moderate-to-large software applications require some form of scheduling, and hence RTOS.

DIFFERENCE: RTOS v/s General Purpose OS

Determinism - The key difference between general-computing operating systems and real-time operating systems is the "deterministic " timing behavior in the real-time operating systems. "Deterministic" timing means that OS consume only known and expected amounts of time. RTOS have their worst case latency defined. Latency is not of a concern for General Purpose OS.

Task Scheduling - General purpose operating systems are optimized to run a variety of applications and processes simultaneously, thereby ensuring that all tasks receive at least some processing time. As a consequence, low-priority tasks may have their priority boosted above other higher priority tasks, which the designer may not want. However, RTOS uses priority-based preemptive scheduling, which allows high-priority threads to meet their deadlines consistently. All system calls are deterministic, implying time bounded operation for all operations and ISRs. This is important for embedded systems where delay could cause a safety hazard. The scheduling in RTOS is time based. In case of General purpose OS, like Windows/Linux, scheduling is process based.

- **Preemptive kernel** - In RTOS, all kernel operations are pre-emptible

- **Priority Inversion** - RTOS have mechanisms to prevent priority inversion

Usage - RTOS are typically used for embedded applications, while General Purpose OS are used for Desktop PCs or other generally purpose PCs.

RTOS CLASSFICATION

RTOS specifies a known maximum time for each of the operations that it performs. Based upon the degree of tolerance in meeting deadlines, RTOS are classified into following categories

- **Hard real-time:** Degree of tolerance for missed deadlines is negligible. A missed deadline can result in catastrophic failure of the system
- **Firm real-time:** Missing a deadly ne might result in an unacceptable quality reduction but may not lead to failure of the complete system
- **Soft real-time:** Deadlines may be missed occasionally, but system doesn't fail and also, system quality is acceptable

For a life saving device, automatic parachute opening device for skydivers, delay can be fatal. Parachute opening device deploys the parachute at a specific altitude based on various conditions. If it fails to respond in specified time, parachute may not get deployed at all leading to casualty. Similar situation exists during inflation of air bags, used in cars, at the time of accident. If airbags don't get inflated at appropriate time, it may be fatal for a driver. So such systems must be hard real time systems, whereas for TV live broadcast, delay can be acceptable. In such cases, soft real time systems can be used.

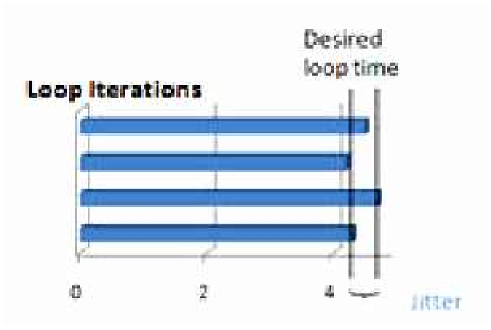


Figure 4.4: - Real-Time Operating System Loop Iteration

Important terminologies used in context of real time systems

- **Determinism:** An application is referred to as deterministic if its timing can be guaranteed within a certain margin of error.
- **Jitter:** Timing error of a task over subsequent iterations of a program or loop is referred to as jitter. RTOS are optimized to minimize jitter.

Check your progress

Q1. What is multiprocessing? Explain its advantages and limitations

Q2. What do you understand by Time-sharing? Also describe the advantages of Time-sharing Systems

Q3. Explain Real time Operating System.

4.4 Summary

In this unit you learnt about functions and types of an operating system like multiprocessing, time sharing and real-time operating system.

- Tightly-coupled systems, all processors share a single system-wide primary memory.
- Loosely-coupled systems, the processors do not share memory, and each processor has its own local memory.
- Multiprocessing is simultaneous execution of two or more processes by a computer system having more than one CPU.
- Time-sharing is a mechanism to provide simultaneous interactive use of a computer system by many users in such a way that all users feel that he/she is the sole user of the system.
- Real-time systems are those systems in which the correctness of the system depends not only on the logical result of computation, but also on the time at which the results are produced.

4.5 Review Questions

Q1. Define the term multiprocessing with example.

Q2. What do you mean by time sharing operating system?

Q3. Define batch processing system in detail.

Q4. Define the types of an operating system.

Q5. What is real time operating system? Explain in details.

BIBLIOGRAPHY

Anderson, R.G. (1900). Data Processing (Vol. 1: Principles and practice; Vol 2: Information Systems and Technology), London: Pitman.

Bingham, J. (1989) Data Processing. Basingstoke: Macmillan.

Bissmer, R.H. (1993). Introduction to Computer Concepts. New York: John Wiley.

IGNOU. CIC-02: The Technology. Block 1 (Hardware). New Delhi: IGNOU.

Jain, V.K. (2000). 'O' Level - Information Technology. New Delhi: BPB.

Rajaraman, V (1997). Fundamentals of Computers. 2nd ed. New Delhi: Prentice- Hall of India.

Rowley (1997). Electronic Library. 4th ed. London: Library Association.

Jacob, Nielsen (1990). *Hypertext and hypermedia*. Boston: Academic Press.

Jain, Satish (1999). *Information technology*. New Delhi: BPB Publications.

Jaiswal, S. (2000). *Information technology today*. New Delhi: Galgotia Pub.

Mahapatra, M and Ramesh D.B. (eds.) (2004), *Information technology application in libraries: a textbook for beginners*. Bhubaneswar: Reprint.

Prasad, A.R.D. (1996). *Some reflection on the impact of IT on LIS profession*. In: *Advances in information technology: impact on library and information field*, edited by I. K. Ravichandra Rao and A.R.D. Prasad. (Proceedings of the 30th DRTC Workshop, 28-30 Oct, 1996, Banagalore). Bangalore: Documentation Research and Training Centre. Paper AC.

Prasad, A.R.D. (1998). *Browsers: the interface to the web*. In: *Practical orientation to Internet*, edited by Devika P. Madalli (Course material of the 31s1 DRTC Workshop, 28-30 Jan, 1998, Bangalore). Bangalore: DRTC and NISSAT. Paper BB (pp 1-8).

Townsend, K and Tapehouse, K (1987). Word processing and desktop publishing. *Communication Technology Impact*, 9(7), 1-4.

Tuck, B (1989). Desktop publishing: what it is and what it can do for you. *Aslib Proceedings*, 41(1), 29-37.

Deital, Harvey M. *Introduction to Operating System*. Warsaw: Addision Wesely Publishing.

Madwick, Stuart E. and Donovan, John J. *Operating System: Concept and Design*. New York: McGraw Hill Ints. Edition.

Silberschatz, Abraham and Peterson, James L. *Operating System Concepts. Warsaw:* Addision Wisely Publishing.

Tanenbaum, Andrew S. and Woodhull, Albert S. (2000). *Operating Systems Design and Implementation:* New Delhi: Prentice Hall of India.