Department of Computer Science
Yusuf Maitama Sule University, Kano

Introduction to Computer Science (CSC1201) Lecture Note

Table of Contents

CHAPTER ONE

COMPUTER EVOLUTION AND PERFORMANCE

1.1 Introduction

We begin our study of computers with a brief history. This history is itself interesting and serves the purpose of providing an overview of computer structure and function. Next, we address the issue of performance. The evolution of computers has been characterized by increasing processor speed, decreasing component size, increasing memory size, and increasing I/O capacity and speed. One factor responsible for the great increase in processor speed is the shrinking size of microprocessor components; this reduces the distance between components and hence increases speed

1.2 History of Computers

1.2.1 First Generation: Vacuum Tubes

ENIAC The ENIAC (Electronic Numerical Integrator And Computer), designed and constructed at the University of Pennsylvania, was the world's first general purpose electronic digital computer. The project was a response to U.S. needs during World War II. The Army's Ballistics Research Laboratory (BRL), an agency responsible for developing range and trajectory tables for new weapons, was having difficulty supplying these tables accurately and within a reasonable time frame. Without these firing tables, the new weapons and artillery were useless to gunners. The BRL employed more than 200 people who, using desktop calculators, solved the necessary ballistics equations. Preparation of the tables for a single weapon would take one person many hours, even days.

John Mauchly, a professor of electrical engineering at the University of Pennsylvania, and John Eckert, one of his graduate students, proposed to build a general-purpose computer using vacuum tubes for the BRL's application. In 1943, the Army accepted this proposal, and work began on the ENIAC. The resulting machine was enormous, weighing 30 tons, occupying 1500 square feet of floor space, and containing more than 18,000 vacuum tubes. When operating, it consumed 140 kilowatts of power. It was also substantially faster than any electromechanical computer, capable of 5000 additions per second.

The ENIAC was a decimal rather than a binary machine. That is, numbers were represented in decimal form, and arithmetic was performed in the decimal system. Its memory consisted of 20 "accumulators," each capable of holding a 10-digit decimal number. A ring of 10 vacuum tubes represented each digit. At any time, only one vacuum tube was in the ON state, representing one of the 10 digits. The major drawback of the ENIAC was that it had to be programmed manually by setting switches and plugging and unplugging cables.

The ENIAC was completed in 1946, too late to be used in the war effort. Instead, its first task was to perform a series of complex calculations that were used to help determine the feasibility of the hydrogen bomb. The use of the ENIAC for a purpose other than that for which it was built demonstrated its general-purpose nature. The ENIAC continued to operate under BRL management until 1955, when it was disassembled.

THE VON NEUMANN MACHINE The task of entering and altering programs for the ENIAC was extremely tedious. The programming process could be facilitated if the program could be represented in a form suitable for storing in memory alongside the data. Then, a computer could get its instructions by reading them from memory, and a program could be set or altered by setting the values of a portion of memory. This idea, known as the stored-program concept, is usually attributed to the ENIAC designers, most notably the mathematician John von Neumann, who was a consultant on the ENIAC project. Alan Turing developed the idea at about the same time. The first publication of the idea was in a 1945 proposal by von Neumann for a new computer, the **EDVAC** (Electronic Discrete Variable Computer). In 1946, von Neumann and his colleagues began the design of a new stored program computer, referred to as the IAS computer, at the Princeton Institute for Advanced Studies. The IAS computer, although not completed until 1952, is the prototype of all subsequent general-purpose computers. Figure 1.1 shows the general structure of the IAS computer

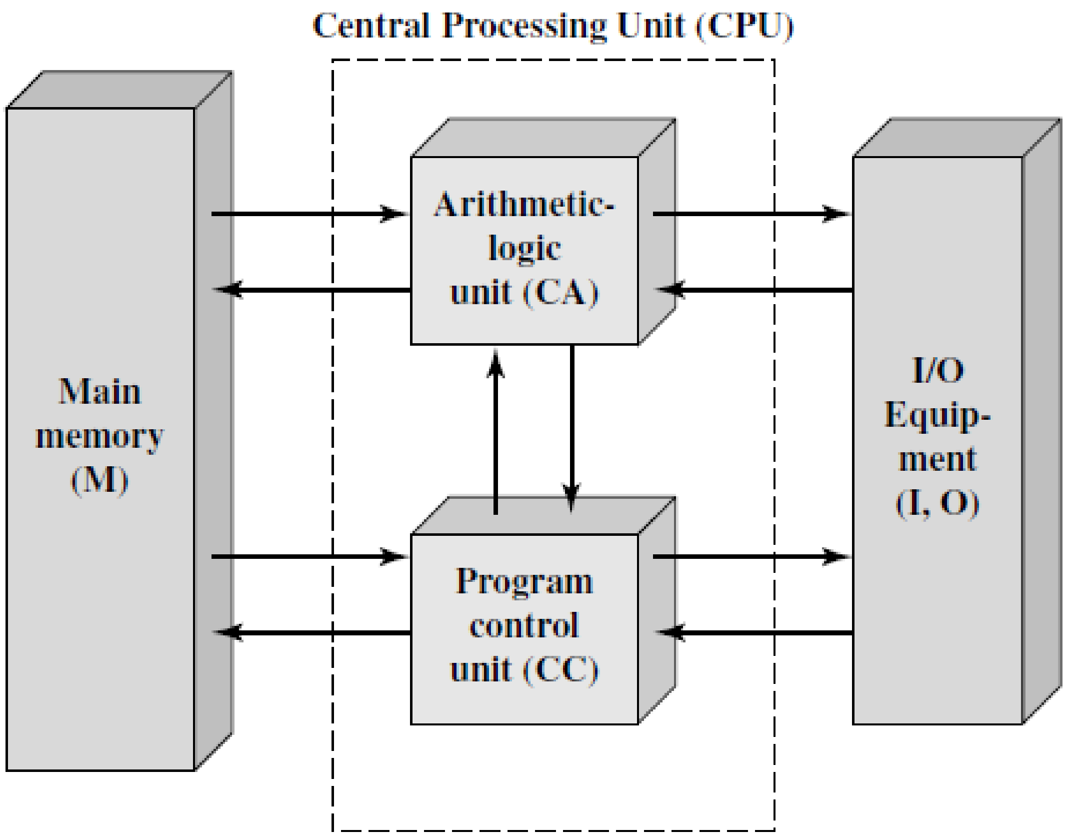

Central Processing Unit (CPU)
Arithmetic-logic unit (CA)
Program control unit (CC)
Main memory (M)
I/O Equip-ment (I, O)

Figure 1.1 Structure of the IAS Computer

It consists of

- A main memory, which stores both data and instructions
- An arithmetic and logic unit (ALU) capable of operating on binary data
- A control unit, which interprets the instructions in memory and causes them to be executed
- Input and output (I/O) equipment operated by the control unit

COMMERCIAL COMPUTERS The 1950s saw the birth of the computer industry with two companies, Sperry and IBM, dominating the marketplace.

In 1947, Eckert and Mauchly formed the Eckert-Mauchly Computer Corporation to manufacture computers commercially. Their first successful machine was the UNIVAC I (Universal Automatic Computer), which was commissioned by the Bureau of the Census for the 1950 calculations. The Eckert-Mauchly Computer Corporation became part of the UNIVAC division of Sperry-Rand Corporation, which went on to build a series of successor machines.

The UNIVAC I was the first successful commercial computer. It was intended for both scientific and commercial applications. The first paper describing the system listed matrix algebraic computations, statistical problems, premium billings for a life insurance company, and logistical problems as a sample of the tasks it could perform.

The UNIVAC II, which had greater memory capacity and higher performance than the UNIVAC I, was delivered in the late 1950s and illustrates several trends that have remained characteristic of the computer industry. First, advances in technology allow companies to continue to build larger, more powerful computers. Second, each company tries to make its new machines backward compatible3 with the older machines. This means that the programs written for the older machines can be executed on the new machine. This strategy is adopted in the hopes of retaining the customer base; that is, when a customer decides to buy a newer machine, he or she is likely to get it from the same company to avoid losing the investment in programs.

The UNIVAC division also began development of the 1100 series of computers, which was to be its major source of revenue. This series illustrates a distinction that existed at one time. The first model, the UNIVAC 1103, and its successors for many years were primarily intended for scientific applications, involving long and complex calculations. Other companies concentrated on business applications, which involved processing large amounts of text data. This split has largely disappeared, but it was evident for several years.

IBM, then the major manufacturer of punched-card processing equipment, delivered its first electronic stored-program computer, the 701, in 1953.The 701 was intended primarily for scientific applications [BASH81]. In 1955, IBM introduced the companion 702 product, which had several hardware features that suited it to business applications. These were the first of a long series of 700/7000 computers that established IBM as the overwhelmingly dominant computer manufacturer.

1.2.2 Second Generation: Transistors

The first major change in the electronic computer came with the replacement of the vacuum tube by the transistor. The transistor is smaller, cheaper, and dissipates less heat than a vacuum tube but can be used in the same way as a vacuum tube to construct computers. Unlike the vacuum tube, which requires wires, metal plates, a glass capsule, and a vacuum, the transistor is a solid-state device, made from silicon.

The transistor was invented at Bell Labs in 1947 and by the 1950s had launched an electronic revolution. It was not until the late 1950s, however, that fully transistorized computers were commercially available. IBM again was not the first company to deliver the new technology.

NCR and, more successfully, RCA were the front-runners with some small transistor machines. IBM followed shortly with the 7000 series.

The use of the transistor defines the second generation of computers. It has become widely accepted to classify computers into generations based on the fundamental hardware technology employed (Table 2.2). Each new generation is characterized by greater processing performance, larger memory capacity, and smaller size than the previous one. But there are other changes as well. The second generation saw the introduction of more complex arithmetic and logic units and control units, the use of high-level programming languages, and the provision of system software with the computer.

The second generation is noteworthy also for the appearance of the Digital Equipment Corporation (DEC). DEC was founded in 1957 and, in that year, delivered its first computer, the PDP-1. This computer and this company began the minicomputer phenomenon that would become so prominent in the third generation.

THE IBM 7094 From the introduction of the 700 series in 1952 to the introduction of the last member of the 7000 series in 1964, this IBM product line underwent an evolution that is typical of computer products. Successive members of the product line show increased performance, increased capacity, and/or lower cost.

1.2.3 Third Generation: Integrated Circuits

A single, self-contained transistor is called a discrete component. Throughout the 1950s and early 1960s, electronic equipment was composed largely of discrete components—transistors, resistors, capacitors, and so on. Discrete components were manufactured separately, packaged in their own containers, and soldered or wired together onto masonite-like circuit boards, which were then installed in computers, oscilloscopes, and other electronic equipment. Whenever an electronic device called for a transistor, a little tube of metal containing a pinhead-sized piece of silicon had to be soldered to a circuit board. The entire manufacturing process, from transistor to circuit board, was expensive and cumbersome.

These facts of life were beginning to create problems in the computer industry. Early second-generation computers contained about 10,000 transistors. This figure grew to the hundreds of thousands, making the manufacture of newer, more powerful machines increasingly difficult.

In 1958 came the achievement that revolutionized electronics and started the era of microelectronics: the invention of the integrated circuit. It is the integrated circuit that defines the third generation of computers. In this section we provide a brief introduction to the technology of integrated circuits. Then we look at perhaps the two most important members of the third generation, both of which were introduced at the beginning of that era: the IBM System/360 and the DEC PDP-8.

MICROELECTRONICS Microelectronics means, literally, “small electronics.” Since the beginnings of digital electronics and the computer industry, there has been a persistent and consistent trend toward the reduction in size of digital electronic circuits. Before examining the implications and benefits of this trend, we need to say something about the nature of digital electronics.

The basic elements of a digital computer, as we know, must perform storage, movement, processing, and control functions. Only two fundamental types of components are required: gates and memory cells. A gate is a device that implements a simple Boolean or logical function, such as IF A AND B ARE TRUE THEN C IS TRUE (AND gate). Such devices are called gates because they control data flow in much the same way that canal gates do. The memory cell is a device that can store one bit of data; that is, the device can be in one of two stable states at any time. By interconnecting large numbers of these fundamental devices, we can construct a computer. We can relate this to our four basic functions as follows:

- Data storage: Provided by memory cells.
- Data processing: Provided by gates.
- Data movement: The paths among components are used to move data from memory to memory and from memory through gates to memory.
- Control: The paths among components can carry control signals. For example, a gate will have one or two data inputs plus a control signal input that activates the gate. When the control signal is ON, the gate performs its function on the data inputs and produces a data output. Similarly, the memory cell will store the bit that is on its input lead when the WRITE control signal is ON and will place the bit that is in the cell on its output lead when the READ control signal is ON.

Thus, a computer consists of gates, memory cells, and interconnections among these elements. The gates and memory cells are, in turn, constructed of simple digital electronic components.

The integrated circuit exploits the fact that such components as transistors, resistors, and conductors can be fabricated from a semiconductor such as silicon. It is merely an extension of the solid-state art to fabricate an entire circuit in a tiny piece of silicon rather than assemble discrete components made from separate pieces of silicon into the same circuit. Many transistors can be produced at the same time on a single wafer of silicon. Equally important, these transistors can be connected with a process of metallization to form circuits.

Initially, only a few gates or memory cells could be reliably manufactured and packaged together. These early integrated circuits are referred to as small-scale integration (SSI). As time went on, it became possible to pack more and more components on the same chip.

IBM SYSTEM/360 By 1964, IBM had a firm grip on the computer market with its 7000 series of machines. In that year, IBM announced the System/360, a new family of computer products. Although the announcement itself was no surprise, it contained some unpleasant news for current IBM customers: the 360 product line was incompatible with older IBM machines. Thus, the transition to the 360 would be difficult for the current customer base. This was a bold step by IBM, but one IBM felt was necessary to break out of some of the constraints of the 7000 architecture and to produce a system capable of evolving with the new integrated circuit technology. The strategy paid off both financially and technically. The 360 was the success of the decade and cemented IBM as the overwhelmingly dominant computer vendor, with a market share above 70%. And, with some modifications and extensions, the architecture of the 360 remains to this day the architecture of IBM's mainframe6 computers. Examples using this architecture can be found throughout this text. The System/360 was the industry's first planned family of computers. The family covered a wide range of performance and cost. Table 2.4 indicates some of the key characteristics of the various models in 1965 (each member of the family is distinguished by a model number). The models were compatible in the sense that a program written for one model should be capable of being executed by another model in the series, with only a difference in the time it takes to execute.

DEC PDP-8 In the same year that IBM shipped its first System/360, another momentous first shipment occurred: PDP-8 from Digital Equipment Corporation (DEC).At a time when the average computer required an air-conditioned room, the PDP-8 (dubbed a minicomputer by the industry, after the miniskirt of the day) was small enough that it could be placed on top of a lab bench or be built into other equipment. It could not do everything the mainframe could, but at $16,000, it was cheap enough for each lab technician to have one. In contrast, the System/360 series of mainframe computers introduced just a few months before cost hundreds of thousands of dollars.

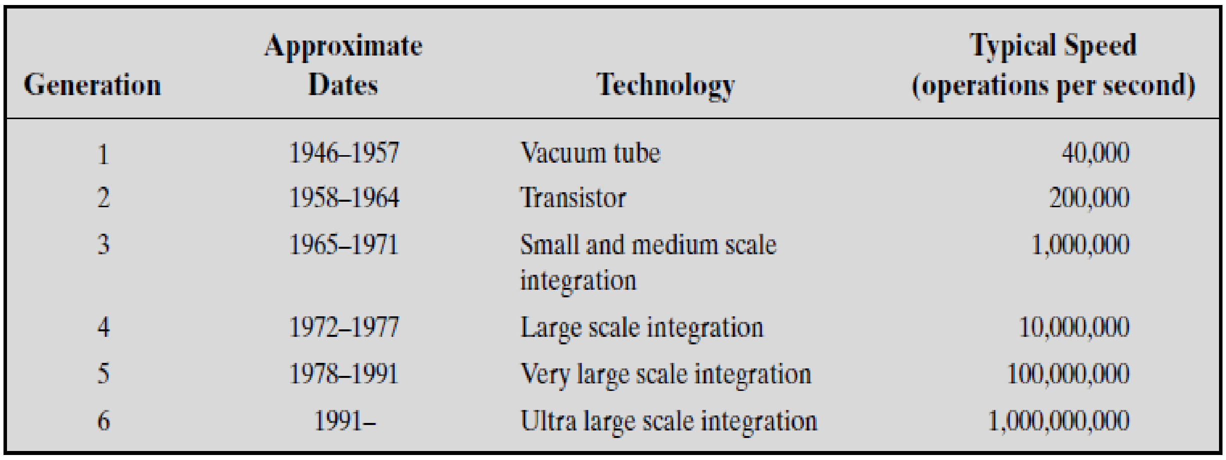
The low cost and small size of the PDP-8 enabled another manufacturer to purchase a PDP-8 and integrate it into a total system for resale. These other manufacturers came to be known as original equipment manufacturers (OEMs), and the OEM market became and remains a major segment of the computer marketplace.

The PDP-8 was an immediate hit and made DEC’s fortune. This machine and other members of the PDP-8 family that followed it (see Table 2.5) achieved a production status formerly reserved for IBM computers, with about 50,000 machines sold over the next dozen years. As DEC’s official history puts it, the PDP-8 “established the concept of minicomputers, leading the way to a multibillion-dollar industry.” It also established DEC as the number one minicomputer vendor, and, by the time the PDP-8 had reached the end of its useful life, DEC was the number two computer manufacturer, behind IBM.

1.2.4 Later Generations

Beyond the third generation there is less general agreement on defining generations of computers. Table 2.2 suggests that there have been a number of later generations, based on advances in integrated circuit technology. With the introduction of largescale integration (LSI), more than 1000 components can be placed on a single integrated circuit chip. Very-large-scale integration (VLSI) achieved more than 10,000 components per chip, while current ultra-large-scale integration (ULSI) chips can contain more than one million components.

Table 1.1 Computer Generations



Generation	Approximate Dates	Technology	Typical Speed (operations per second)
1	1946–1957	Vacuum tube	40,000
2	1958–1964	Transistor	200,000
3	1965–1971	Small and medium scale integration	1,000,000
4	1972–1977	Large scale integration	10,000,000
5	1978–1991	Very large scale integration	100,000,000
6	1991–	Ultra large scale integration	1,000,000,000

With the rapid pace of technology, the high rate of introduction of new products, and the importance of software and communications as well as hardware, the classification by generation becomes less clear and less meaningful. It could be said that the commercial application of new developments resulted in a major change in the early 1970s and that the results of these changes are still being worked out. In this section, we mention two of the most important of these results.

SEMICONDUCTOR MEMORY The first application of integrated circuit technology to computers was construction of the processor (the control unit and the arithmetic and logic unit) out of integrated circuit chips. But it was also found that this same technology could be used to construct memories.

In the 1950s and 1960s, most computer memory was constructed from tiny rings of ferromagnetic material, each about a sixteenth of an inch in diameter. These rings were strung up on grids of fine wires suspended on small screens inside the computer. Magnetized one way, a ring (called a core) represented a one; magnetized the other way, it stood for a zero. Magnetic-core memory was rather fast; it took as little as a millionth of a second to read a bit stored in memory. But it was expensive, bulky, and used destructive readout: The simple act of reading a core erased the data stored in it. It was therefore necessary to install circuits to restore the data as soon as it had been extracted.

Then, in 1970, Fairchild produced the first relatively capacious semiconductor memory. This chip, about the size of a single core, could hold 256 bits of memory. It was nondestructive and much faster than core. It took only 70 billionths of a second to read a bit. However, the cost per bit was higher than for that of core. In 1974, a seminal event occurred: The price per bit of semiconductor memory dropped below the price per bit of core memory. Following this, there has been a continuing and rapid decline in memory cost accompanied by a corresponding increase in physical memory density. This has led the way to smaller, faster machines with memory sizes of larger and more expensive machines from just a few years earlier. Developments in memory technology, together with developments in processor technology to be discussed next, changed the nature of computers in less than a decade. Although bulky, expensive computers remain a part of the landscape, the computer has also been brought out to the "end user," with office machines and personal computers. Since 1970, semiconductor memory has been through 13 generations: 1K, 4K, 16K, 64K, 256K, 1M, 4M, 16M, 64M, 256M, 1G, 4G, and, as of this writing, 16 Gbits on a single chip (1K = 2^{10}, 1M 2^{20}, 1G 2^{30}). Each generation has provided four times the storage density of the previous generation, accompanied by declining cost per bit and declining access time.

MICROPROCESSORS Just as the density of elements on memory chips has continued to rise, so has the density of elements on processor chips. As time went on, more and more elements were placed on each chip, so that fewer and fewer chips were needed to construct a single computer processor. A breakthrough was achieved in 1971, when Intel developed its 4004.The 4004 was the first chip to contain all of the components of a CPU on a single chip: The microprocessor was born.

The 4004 can add two 4-bit numbers and can multiply only by repeated addition. By today's standards, the 4004 is hopelessly primitive, but it marked the beginning of a continuing evolution of microprocessor capability and power.

This evolution can be seen most easily in the number of bits that the processor deals with at a time.There is no clear-cut measure of this, but perhaps the best measure is the data bus width: the number of bits of data that can be brought into or sent out of the processor at a time. Another measure is the number of bits in the accumulator or in the set of general-purpose registers. Often, these measures coincide, but not always. For example, a number of microprocessors were developed that operate on 16-bit numbers in registers but can only read and write 8 bits at a time.

The next major step in the evolution of the microprocessor was the introduction in 1972 of the Intel 8008.This was the first 8-bit microprocessor and was almost twice as complex as the 4004.

Neither of these steps was to have the impact of the next major event: the introduction in 1974 of the Intel 8080.This was the first general-purpose microprocessor. Whereas the 4004 and the 8008 had been designed for specific applications, the 8080 was designed to be the CPU of a general-purpose microcomputer. Like the 8008, the 8080 is an 8-bit microprocessor. The 8080, however, is faster, has a richer instruction set, and has a large addressing capability.

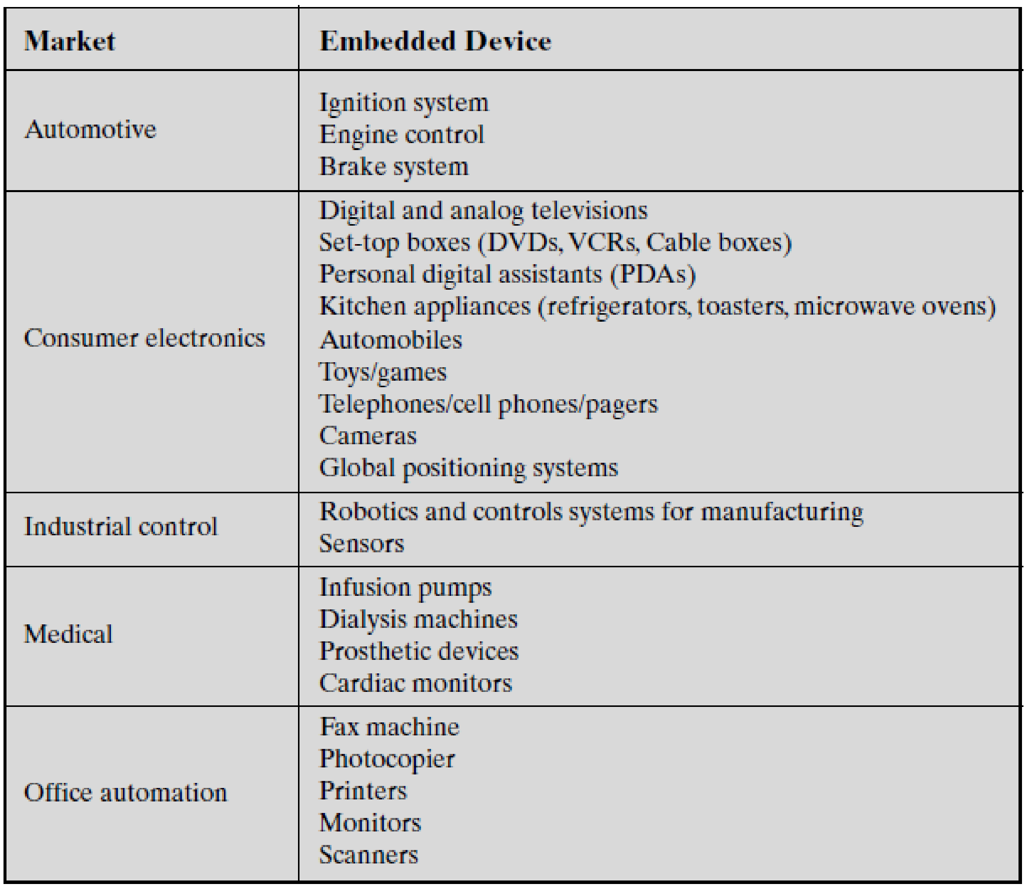
About the same time, 16-bit microprocessors began to be developed. However, it was not until the end of the 1970s that powerful, general-purpose 16-bit microprocessors appeared. One of these was the 8086. The next step in this trend occurred in 1981, when both Bell Labs and Hewlett-Packard developed 32-bit, single-chip microprocessors. Intel introduced its own 32-bit microprocessor, the 80386, in 1985.

Embedded Systems

The term embedded system refers to the use of electronics and software within a product, as opposed to a general-purpose computer, such as a laptop or desktop system. The following is a good general definition: Embedded system. *A combination of computer hardware and software, and perhaps additional mechanical or other parts, designed to perform a dedicated function. In many cases, embedded systems are part of a larger system or product, as in the case of an antilock braking system in a car.*

Embedded systems far outnumber general-purpose computer systems, encompassing a broad range of applications. Table 1.2 lists examples of embedded systems.

Table 1.2 Examples of embedded systems



Market	Embedded Device
Automotive	Ignition system Engine control Brake system
Consumer electronics	Digital and analog televisions Set-top boxes (DVDs, VCRs, Cable boxes) Personal digital assistants (PDAs) Kitchen appliances (refrigerators, toasters, microwave ovens) Automobiles Toys/games Telephones/cell phones/pagers Cameras Global positioning systems
Industrial control	Robotics and controls systems for manufacturing Sensors
Medical	Infusion pumps Dialysis machines Prosthetic devices Cardiac monitors
Office automation	Fax machine Photocopier Printers Monitors Scanners

These systems have widely varying requirements and constraints, such as the following:

- Small to large systems, implying very different cost constraints, thus different needs for optimization and reuse
- Relaxed to very strict requirements and combinations of different quality requirements, for example, with respect to safety, reliability, real-time, flexibility, and legislation
- Short to long lifetimes
- Different environmental conditions in terms of, for example, radiation, vibrations, and humidity
- Different application characteristics resulting in static versus dynamic loads, slow to fast speed, compute versus interface intensive tasks, and/or combinations thereof
- Different models of computation ranging from discrete-event systems to those involving continuous time dynamics (usually referred to as hybrid systems)

Often, embedded systems are tightly coupled to their environment. This can give rise to real-time constraints imposed by the need to interact with the environment. Constraints, such as required speeds of motion, required precision of measurement, and required time durations, dictate the timing of software operations.

1.3 Performance Assessments of Processor

We examine the most common approach to assessing processor and computer system performance in the following subsections.

1.3.1 Clock Speed and Instructions per Second

THE SYSTEM CLOCK Operations performed by a processor, such as fetching an instruction, decoding the instruction, performing an arithmetic operation, and so on, are governed by a system clock. Typically, all operations begin with the pulse of the clock. Thus, at the most fundamental level, the speed of a processor is dictated by the pulse frequency produced by the clock, measured in cycles per second, or Hertz (Hz).

Typically, clock signals are generated by a quartz crystal, which generates a constant signal wave while power is applied. This wave is converted into a digital voltage pulse stream that is provided in a constant flow to the processor circuitry. For example, a 1-GHz processor receives 1 billion pulses per second. The rate of pulses is known as the clock rate, or clock speed. One increment, or pulse, of the clock is referred to as a clock cycle, or a clock tick. The time between pulses is the cycle time. The clock rate is not arbitrary but must be appropriate for the physical layout of the processor. Actions in the processor require signals to be sent from one processor element to another. When a signal is placed on a line inside the processor.

INSTRUCTION EXECUTION RATE A processor is driven by a clock with a constant frequency *f* or, equivalently, a constant cycle time τ, where $\tau = {}^{1}/_{f}$. Define the instruction count, I_c, for a program as the number of machine instructions executed for that program until it runs to completion or for some defined time interval. Note that this is the number of instruction executions, not the number of instructions in the object code of the program. An important parameter is the average cycles per instruction CPI for a program. If all instructions required the same number of clock cycles, then CPI would be a constant value for a processor. However, on any give processor, the number of clock cycles required varies for different types of instructions, such as load, store, branch, and so on. Let CPI_i be the number of cycles required for instruction type *i*. and I_i be the number of executed instructions of type *I* for a given program. Then we can calculate an overall CPI as follows:

$$CPI = \frac{\sum_i^n (CPI_i \times I_i)}{I_c}$$

The processor time *T* needed to execute a given program can be expressed as

$$T = I_c \times CPI \times \tau$$

We can refine this formulation by recognizing that during the execution of an instruction, part of the work is done by the processor, and part of the time a word is being transferred to or from memory. In this latter case, the time to transfer depends on the memory cycle time, which may be greater than the processor cycle time. We can rewrite the preceding equation as

$$T = I_c \times [p + (m \times k)] \times \tau$$

where *p* is the number of processor cycles needed to decode and execute the instruction, *m* is the number of memory references needed, and *k* is the ratio between memory cycle time and processor cycle time. The five performance factors in the preceding equation (I_c, p, m, k, τ) are influenced by four system attributes: the design of the instruction set (known as instruction set architecture), compiler technology (how effective the compiler is in producing an efficient machine language program from a high-level language program), processor implementation, and cache and memory hierarchy.

A common measure of performance for a processor is the rate at which instructions are executed, expressed as millions of instructions per second (MIPS), referred to as the MIPS rate. We can express the MIPS rate in terms of the clock rate and CPI as follows:

$$\text{MIPS rate} = \frac{I_c}{T \times 10^6} = \frac{f}{CPI \times 10^6}$$

CHAPTER TWO

SOME BASIC TERMS

Here some of the ideas and terminology of computer is given below.

- **Data:** Basically data is a collection of facts from which information may be derived. Data is defined as an un-processed collection of raw facts in a manner suitable for communication, interpretation or processing.
- **Computer:** A computer is a device that works under the control of stored *programs*, automatically accepting, storing and *processing* data to produce *information* that is the result of the processing.

When a computer processes data it actually performs a number of separate functions as follows:

- **Input:** allows the user to enter the program and data and send it to the processing unit. The common input devices are keyboard, mouse and scanners.
- **Storage**: usually means secondary storage, which stores data and programs. Here the data and programs are permanently stored for future use.
- **Processing:** a computer performs operations on the data it holds within.
- **Output:** a computer produces results of its processing for external use.
- **Program:** A program is a set of instructions that tells the computer exactly how to manipulate the input data and produce the desired output.
- **Information:** A distinction is sometimes made between data and information. When data is converted into a more useful or intelligible form then it is said to be processed into information.
- **Hardware:** is the general term used to describe all the electronic and mechanical elements of the computer, together with those devices used with the computer.
- **Software:** is the general term used to describe all the various programs that may be used on a computer system together with their associated documentation.
- **Bit (Binary Digit):** The smallest unit of information storable in a computer, expressed as 0 or 1.
- **Byte (Binary Digit Eight):** A set of 8 adjacent bits, which represent a unit of computer memory equal to that needed to store a single character.
- **Memory:** Physical device that is used to store such information as data or program on a temporary or permanent basis for use in a computer.
- **Semi-conductor Memory:** Any class of computer memory devices consisting of one or more integrated circuits.
- **Random Access Memory** (RAM) is the most common type of memory found in the modern computers. This is really the main store and is the place where the program gets stored. When the CPU runs a program, it fetches the program instructions from the RAM

and carries them out. If the CPU needs to store the results of the calculations it can store them in RAM. When we switch off a computer, whatever is stored in the RAM gets erased. It is a volatile form of memory.

- **Read Only Memory (ROM):** In ROM, the information is burnt (pre-recorded) into the ROM chip at manufacturing time. Once data has been written into a ROM chip, it cannot be erased but you can read it. When we switch off the computer, the contents of the ROM are not erased but remain stored permanently. ROM is a non-volatile memory. ROM stores critical programs such as the program that boots the computer.

2.1 Computer Organization

Virtually every computer regardless of difference in physical appearance can be divided into logical units, or sections:

2.1.1 Input Unit

Input devices allow the user to input data and instructions to the computer. There are a variety of input devices. Direct entry of data generally requires a keyboard. It may also use other devices for direct data entry like a touch sensitive screen, voice recognition system and scanners. The popular input devices are discussed below.

2.1.2 Output Unit

This is the *shipping* section of the computer that takes information that the computer has processed and placed it on various output devices, like monitor, printer, and speaker, making the information available for use outside the computer. Computers can output information in various ways, including displaying the output on screens, playing it on audio/video devices, printing it on paper or using the output to control other devices.

2.1.3 Memory Unit

This is the rapid-access, relatively low-cost *storage house* section of the computer, it facilitates the temporary storage of data. The memory unit retains information that has been entered through the input unit, enabling that information to be immediately available for processing. In addition, the unit retains processed information until that information can be transmitted to output devices. Often the memory unit is called either *memory* or *primary memory – random access memory* (*RAM*) is an example of primary memory. Primary memory is usually volatile, which means that it is erased when the machine is powered off.

2.1.4 Secondary Storage Unit

This unit is the long-term, high capacity *storage housing* of the computer. Secondary storage devices, such as hard drives and disks, normally hold programs or data that other units are not actively using; the computer then can retrieve this information when it is needed – hours, days, months or even years later. Information in secondary storage takes much longer to access than information in primary storage. However, the price per unit of secondary storage is much less than the price per unit of primary memory. Secondary storage is usually non-volatile – it retains information even when the computer is switched off. It comes in many forms such as the magnetic disk.

2.2 Central Processing Unit (CPU)

The CPU serves as the *administrative* section of the computer. This is the computer's coordinator, responsible for supervising the operation of the other sections. The CPU alerts the input unit when information should be read into the memory unit, instructs the ALU about when to use information from the memory unit in calculations and tells the output unit when to send the information from the memory unit to certain output devices. Sometimes the ALU & CPU are regarded as a single unit.

2.2.1 Arithmetic Logic Unit (ALU)

The ALU is the *manufacturing* section of the computer. It is responsible for the performance of calculations such as addition, subtraction, multiplication and division. It also contains decision mechanisms, allowing the computer to perform such tasks as determining whether two items stored in memory are equal.

2.2.2 Control Unit (CU)

The control unit directs and controls the activities of the computer system. It interprets the instructions fetched from the main memory of the computer, sends the control signals to the devices involved in the execution of the instructions.

The various units of a computer system are shown in figure 2.1

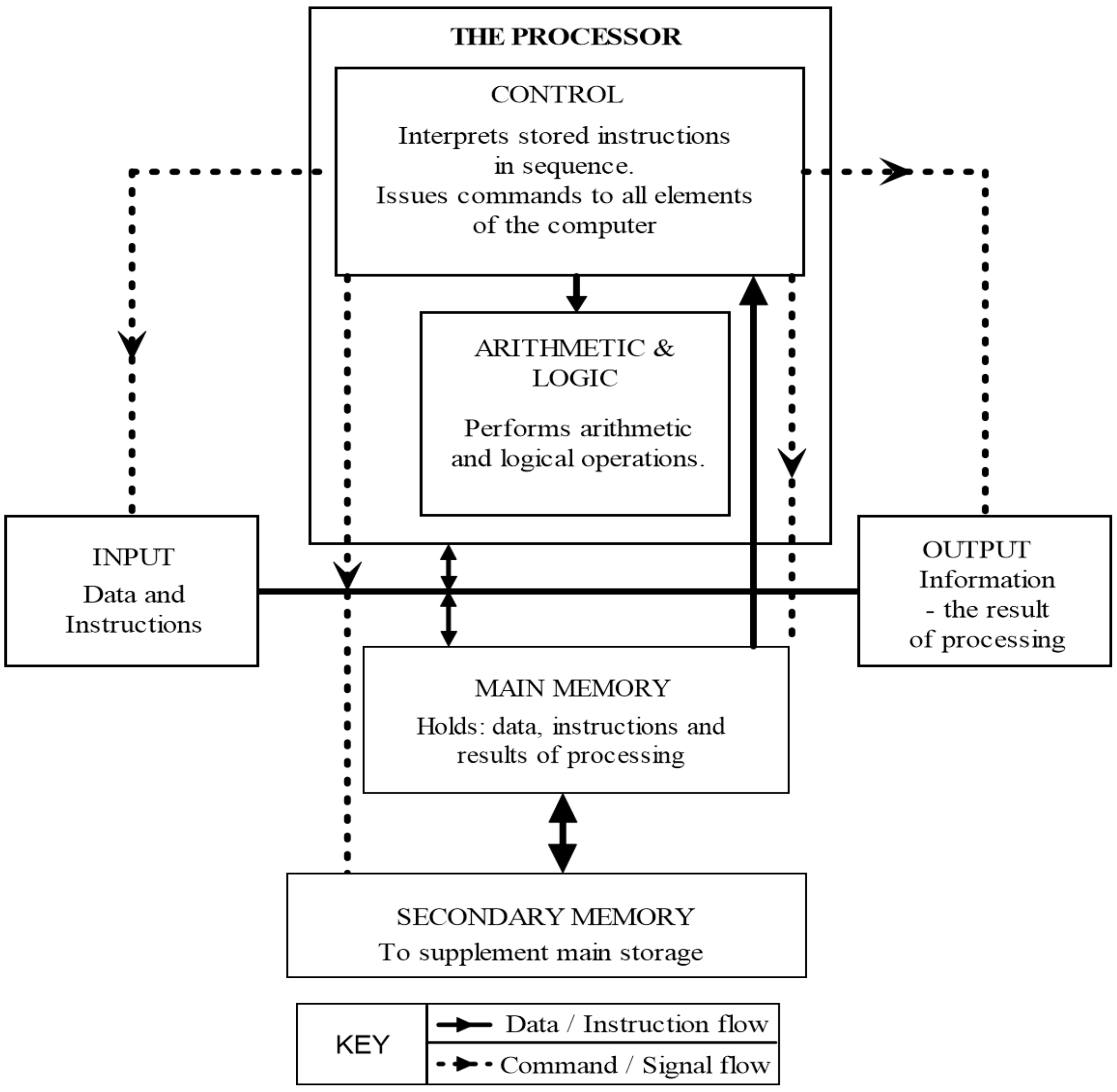

THE PROCESSOR
CONTROL
Interprets stored instructions in sequence.
Issues commands to all elements of the computer
ARITHMETIC & LOGIC
Performs arithmetic and logical operations.
INPUT
Data and Instructions
OUTPUT
Information - the result of processing
MAIN MEMORY
Holds: data, instructions and results of processing
SECONDARY MEMORY
To supplement main storage
KEY
Data / Instruction flow
Command / Signal flow

figure 2.1 Components of a Computer System

CHAPTER THREE

THE COMPUTER SYSTEM

3.1 Introduction

At a top level, a computer consists of CPU (central processing unit), memory, and I/O components, with one or more modules of each type. These components are interconnected in some fashion to achieve the basic function of the computer, which is to execute programs. Thus, at a top level, we can describe a computer system by (1) describing the external behavior of each component – that is, the data and control signals that it exchanges with other components; and (2) describing the interconnection structure and the controls required to manage the use of the interconnection structure.

This top-level view of structure and function is important because of its explanatory power in understanding the nature of a computer. Equally important is its use to understand the increasingly complex issues of performance evaluation. A grasp of the top-level structure and function offers insight into system bottlenecks, alternate pathways, the magnitude of system failures if a component fails, and the ease of adding performance enhancements. In many cases, requirements for greater system power and fail-safe capabilities are being met by changing the design rather than merely increasing the speed and reliability of individual components.

3.2 Computer Components

virtually all contemporary computer designs are based on concepts developed by John von Neumann at the Institute for Advanced Studies, Princeton. Such a design is referred to as the von Neumann architecture and is based on three key concepts:

- Data and instructions are stored in a single read–write memory.
- The contents of this memory are addressable by location, without regard to the type of data contained there.
- Execution occurs in a sequential fashion (unless explicitly modified) from one instruction to the next.

The reasoning behind these concepts summarized here. There is a small set of basic logic components that can be combined in various ways to store binary data and to perform arithmetic and logical operations on that data. If there is a computation to be performed, a configuration of logic components designed specifically for that computation could be constructed. We can think of the process of connecting the various components in the desired configuration as a form of programming. The resulting “program” is in the form of hardware and is termed a *hardwired program*.

Now consider this alternative. Suppose we construct a general-purpose configuration of arithmetic and logic functions. This set of hardware will perform various functions on data depending on control signals applied to the hardware. In the original case of customized hardware, the system accepts data and produces results (Figure 3.1a). With general-purpose hardware, the system accepts data and control signals and produces results. Thus, instead of rewiring the hardware for each new program, the programmer merely needs to supply a new set of control signals.

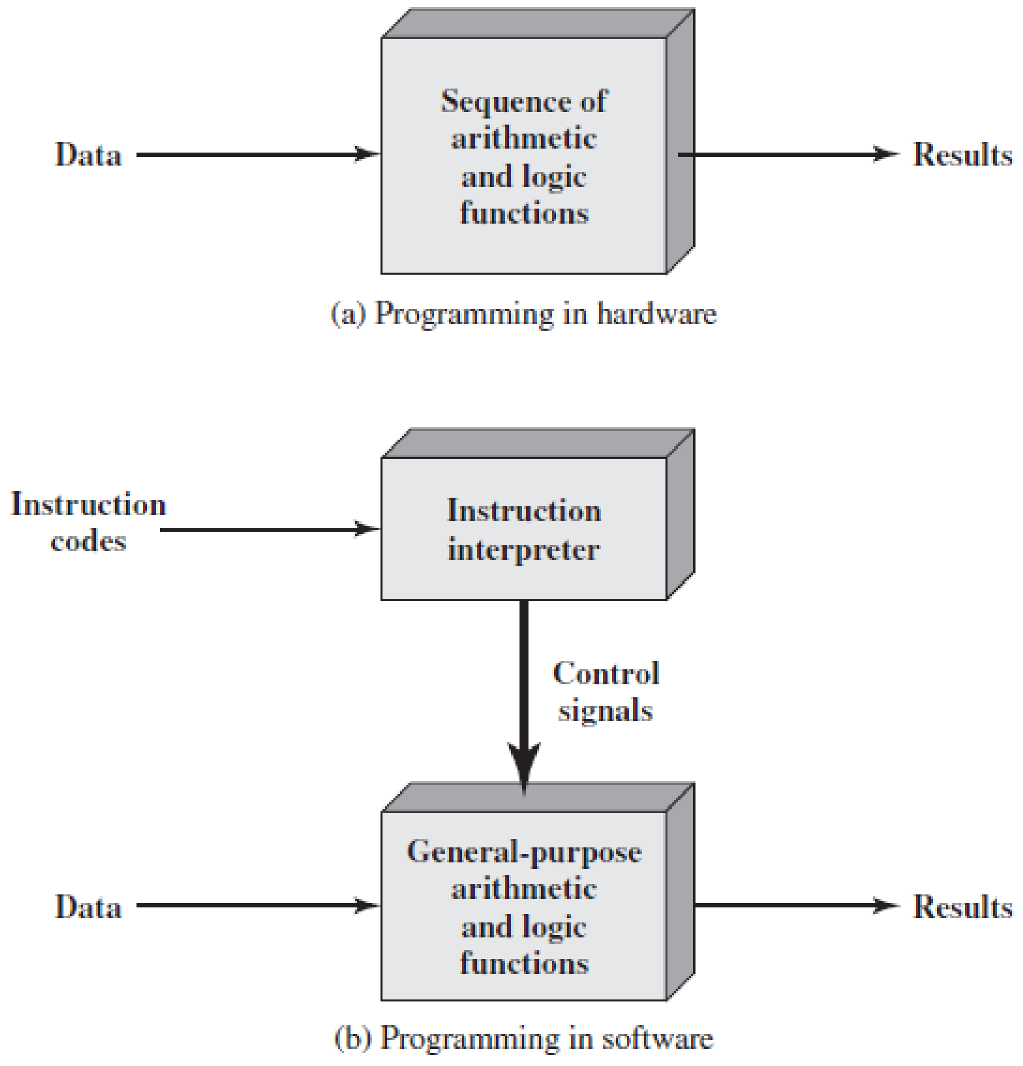

Data
Sequence of arithmetic and logic functions
Results
(a) Programming in hardware
Instruction codes
Instruction interpreter
Control signals
Data
General-purpose arithmetic and logic functions
Results
(b) Programming in software

Figure 3.1 Hardware and Software Approaches.

How shall control signals be supplied? The answer is simple but subtle. The entire program is a sequence of steps. At each step, some arithmetic or logical operation is performed on some data. For each step, a new set of control signals is needed. Let us provide a unique code for each possible set of control signals, and let us add to the general-purpose hardware a segment that can accept a code and generate control signals (Figure 3.1b).

Programming is now much easier. Instead of rewiring the hardware for each new program, all we need to do is provide a new sequence of codes. Each code is, in effect, an instruction, and part of the hardware interprets each instruction and generates control signals. To distinguish this new method of programming, a sequence of codes or instructions is called *software*.

Figure 2.1b indicates two major components of the system: an instruction interpreter and a module of general-purpose arithmetic and logic functions. These two constitute the CPU. Several other components are needed to yield a functioning computer. Data and instructions must be put into the system. For this we need some sort of input module. This module contains basic components for accepting data and instructions in some form and converting them into an internal form of signals usable by the system. A means of reporting results is needed, and this is in the form of an output module. Taken together, these are referred to as I/O components.

One more component is needed. An input device will bring instructions and data in sequentially. But a program is not invariably executed sequentially; it may jump around (e.g., the IAS jump instruction). Similarly, operations on data may require access to more than just one element at a time in a predetermined sequence. Thus, there must be a place to store temporarily both instructions and data. That module is called memory, or main memory to distinguish it from external storage or peripheral devices. Von Neumann pointed out that the same memory could be used to store both instructions and data.

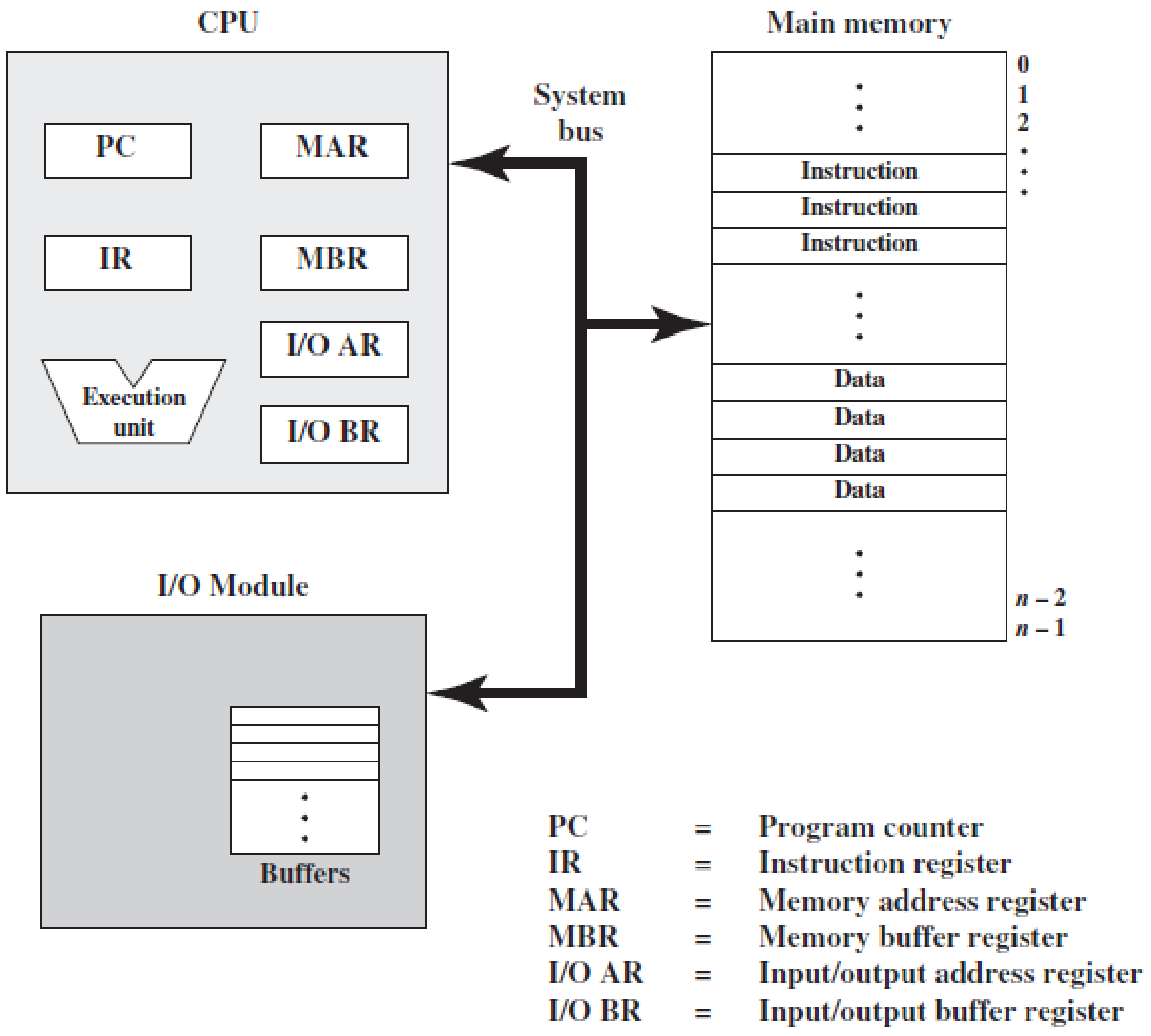


Figure 3.2 Top-Level View of Computer Components

Figure 3.2 illustrates these top-level components and suggests the interactions among them. The CPU exchanges data with memory. For this purpose, it typically makes use of two internal (to the CPU) registers: a memory address register (MAR), which specifies the address in memory for the next read or write, and a memory buffer register (MBR), which contains the data to be written into memory or receives the data read from memory.

Similarly, an I/O address register (I/OAR) specifies a particular I/O device. An I/O buffer (I/OBR) register is used for the exchange of data between an I/O module and the CPU.

A memory module consists of a set of locations, defined by sequentially numbered addresses. Each location contains a binary number that can be interpreted as either an instruction or data. An I/O module transfers data from external devices to CPU and memory, and vice versa. It contains internal buffers for temporarily holding these data until they can be sent on.

Having looked briefly at these major components, we now turn to an overview of how these components function together to execute programs.

3.3 Function of a Computer

The basic function performed by a computer is execution of a program, which consists of a set of instructions stored in memory. The processor does the actual work by executing instructions specified in the program. This section provides an overview of the key elements of program execution. In its simplest form, instruction processing consists of two steps: The processor reads (fetches) instructions from memory one at a time and executes each instruction. Program execution consists of repeating the process of instruction fetch and instruction execution. The instruction execution may involve several operations and depends on the nature of the instruction.

The processing required for a single instruction is called an *instruction cycle*. Using the simplified two-step description given previously, the instruction cycle is depicted in Figure 3.3.

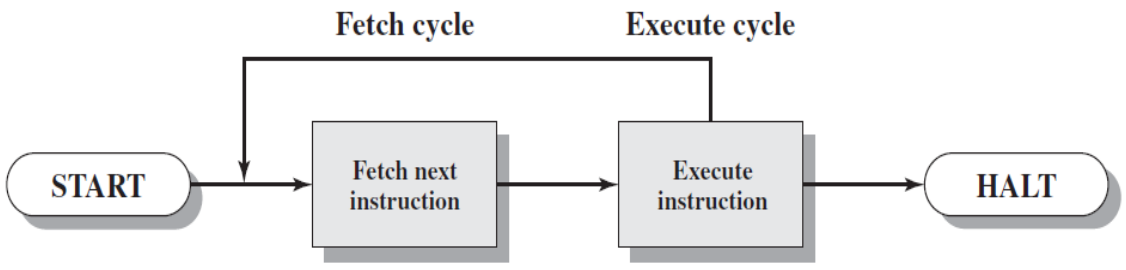


Figure 3.3 Instruction Cycle

The two steps are referred to as the fetch cycle and the execute cycle. Program execution halts only if the machine is turned off, some sort of unrecoverable error occurs, or a program instruction that halts the computer is encountered.

3.3.1 Instruction Fetch and Execute

At the beginning of each instruction cycle, the processor fetches an instruction from memory. In a typical processor, a register called the program counter (PC) holds the address of the instruction to be fetched next. Unless told otherwise, the processor always increments the PC after each instruction fetch so that it will fetch the next instruction in sequence (i.e., the instruction located at the next higher memory address). So, for example, consider a computer in which each instruction occupies one 16-bit word of memory. Assume that the program counter is set to location 300.The processor will next fetch the instruction at location 300. On succeeding instruction cycles, it will fetch instructions from locations 301, 302, 303, and so on. This sequence may be altered, as explained presently.

The fetched instruction is loaded into a register in the processor known as the instruction register (IR). The instruction contains bits that specify the action the processor is to take. The processor interprets the instruction and performs the required action. In general, these actions fall into four categories:

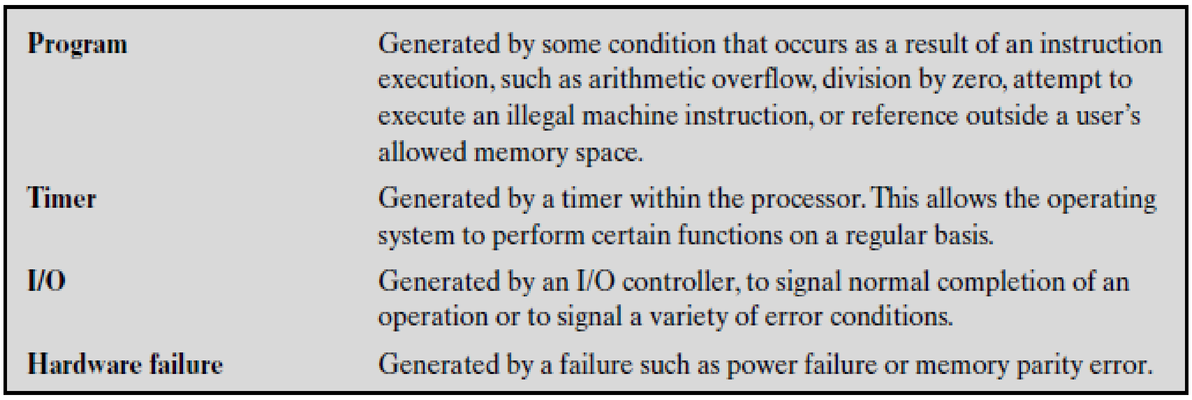
- **Processor-memory:** Data may be transferred from processor to memory or from memory to processor.
- **Processor-I/O:** Data may be transferred to or from a peripheral device by transferring between the processor and an I/O module.
- **Data processing:** The processor may perform some arithmetic or logic operation on data.
- **Control:** An instruction may specify that the sequence of execution be altered. For example, the processor may fetch an instruction from location 149, which specifies that the next instruction be from location 182. The processor will remember this fact by setting the program counter to 182.Thus, on the next fetch cycle, the instruction will be fetched from location 182 rather than 150.

An instruction's execution may involve a combination of these actions.

3.3.2 Interrupts

Virtually all computers provide a mechanism by which other modules (I/O, memory) may interrupt the normal processing of the processor. Table 3.1 lists the most common classes of interrupts.

Table 3.1 Classes of Interrupts



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

However, we need to introduce the concept now to understand more clearly the nature of the instruction cycle and the implications of interrupts on the interconnection structure. The reader need not be concerned at this stage about the details of the generation and processing of interrupts, but only focus on the communication between modules that results from interrupts.

Interrupts are provided primarily as a way to improve processing efficiency. For example, most external devices are much slower than the processor. Suppose that the processor is transferring data to a printer using the instruction cycle scheme of Figure 3.3. After each write operation, the processor must pause and remain idle until the printer catches up. The length of this pause may be on the order of many hundreds or even thousands of instruction cycles that do not involve memory. Clearly, this is a very wasteful use of the processor.

3.3.3 I/O Function

Thus far, we have discussed the operation of the computer as controlled by the processor, and we have looked primarily at the interaction of processor and memory. The discussion has only alluded to the role of the I/O component.

An I/O module (e.g., a disk controller) can exchange data directly with the processor. Just as the processor can initiate a read or write with memory, designating the address of a specific location, the processor can also read data from or write data to an I/O module. In this latter case, the processor identifies a specific device that is controlled by a particular I/O module. Thus, an instruction sequence could occur, with I/O instructions rather than memory referencing instructions.

In some cases, it is desirable to allow I/O exchanges to occur directly with memory. In such a case, the processor grants to an I/O module the authority to read from or write to memory, so that the I/O-memory transfer can occur without tying up the processor. During such a transfer, the I/O module issues read or write commands to memory, relieving the processor of responsibility for the exchange. This operation is known as direct memory access (DMA).

3.4 Interconnection Structures

A computer consists of a set of components or modules of three basic types (processor, memory, I/O) that communicate with each other. In effect, a computer is a network of basic modules. Thus, there must be paths for connecting the modules.

The collection of paths connecting the various modules is called the *interconnection structure*. The design of this structure will depend on the exchanges that must be made among modules. The main modules of a computer system are depicted in figure 3.4

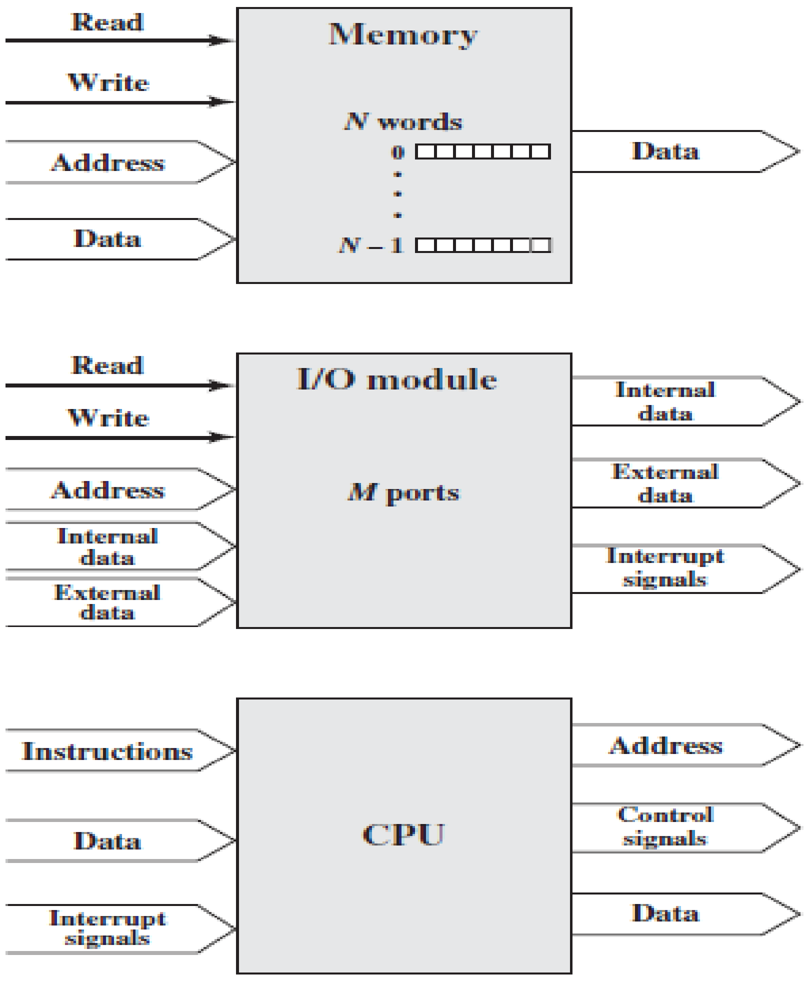


Figure 3.4 Computer Modules

CHAPTER FOUR

CLASSIFICATION OF COMPUTERS

There are several methods of classifying computers. First the main distinction between digital and analog devices is given, followed by classification of decreasing power and size, and finally classification by age of technology is given.

4.1 Classification Based on the Type of Data they Process

This classification is more pronounced in the early days of modern electronic computer. But nowadays, majority of the computers can virtually process all forms of data; they are hybrid in nature

4.1.1 Digital Computer

The word "digital" as used here means whole numbers (discrete). A digital computer is computer that stores and performs a series of mathematical and logical operations on data expressed as discrete signals interpreted as numbers, usually in the form of binary notation.

4.1.2 Analog Computer

Analog Computer are akin to measuring instruments such as thermometers and voltmeters with pointers and dials. They process data in the form of electric voltages, which are variable positions of a pointer on a dial. The output from analog computers is often in the form of smooth graphs from which information can be read.

4.1.3 Hybrid Computer

A hybrid computing system is a combination of desirable features of analog and digital computers. It is mostly used for automatic operations of complicated physical processes and machines. Now-a-days analog-to-digital and digital-to-analog converters are used for transforming the data into suitable form for either type of computation.

4.2 Classification Based on Processing Power and Size

The following classification is in order of decreasing power and size. However, there are no sharp dividing lines in that, for example, a model at the top of a manufacturer's range of minicomputers might well be more powerful than the model at the bottom of a range of mainframes.

4.2.1 Supercomputers

These are the largest, fastest and most expensive computers. They are very powerful computers that have extremely fast processors, capable of performing 30-50 Gflops, i.e. 30–50 billion floating–point operations per second. They are very expensive (several million dollars) and are typically used for the toughest computational tasks. Supercomputers are used, for example, in meteorology, engineering, nuclear physics, and astronomy. Several hundred are in operation worldwide at present. Examples of supercomputers include the *CRAY-MP* and *CRAY*

2 supercomputers which cost over $5 million each. Principal manufactures are Cray Research and NEC, Fujitsu, and Hitachi of Japan.

4.2.2 Mainframes

These are large general-purpose computers with extensive processing, storage and input/output capabilities. They are used in centralized computing environment and normally data input is achieved via terminals wired to the mainframe computer. Mainframe computers usually need a specialized environment in which to operate – with dust, temperature and humidity carefully controlled. Mainframes are usually owned by giant corporate organizations, such as universities, research institutes, giant banks, etc. Mainframes are usually sophisticate and large; thus they call for great detail of support from their manufacturers and representatives. Example of mainframes are *IBM 360/370* system and *NCR V-8800* system. The market of mainframes is dominated by IBM.

4.2.3 Minicomputers (mini)

This is a name originally given to computers that physically went within a single equipment cabinet, i.e. on the order of a few cubic feet. Compared with large computers, minicomputers were cheaper and smaller, with smaller memory. The word minicomputer is no longer used very specifically, it predates the term microcomputer and a boundary between these two classes of devise is unclear. Examples of minicomputers are *PDP II*, *VAX 750/6000, NCR 9300, DEC, HP 3000, IBM system 38* and *MV400.*

4.2.4 Microcomputers

These are computer systems that utilize a microprocessor as their central and arithmetic element. The personal computer (PC) is one form. The power and price of a microcomputer is determine partly by the speed and power of the processor and partly by the characteristics of other computers of the system, i.e. the memory, the disk units, the display, the keyboard, the flexibility of the hardware, and the operating system and other software. Examples include IBM PC and its compatibles and Apple Macintosh.

CHAPTER FIVE

HARDWARE AND SOFTWARE

5.1 Hardware

Computer hardware consists of the components that can be physically handled. The function of these components is basically divided into four main categories: input, processing, storage and output services. The functions of these units were already discussed in section 4.

The four units are interdependent (i.e. the function of one unit depends on the function of the other.) They interact harmoniously to provide the full function of the computer's hardware. The units connect to the microprocessors, specifically the computer's central processing unit (CPU) – the electronic circuitry that provides the computational ability and control of the computer.

For most computers, the principal input device is the keyboard. Storage devices include external floppy disks and internal hard disks, output devices that display data include monitors and printers

5.1.1 Peripherals

Peripheral is a term used for devices such as disk drives, printers, modems, and joysticks, that are connected to a computer and are controlled by its microprocessor. Although peripheral often implies "additional but not essential", many peripheral devices are critical elements of a fully functioning and useful computer system. Few people, for example, would argue that disk drives are nonessential, although computers can function without them.

Keyboards, monitors and mice are also strictly considered peripheral devise, but because they represent primary source of input and output in most computer systems, they can be considered more as extension of the system unit than as peripherals.

Keyboard: (*input*) the most common input device is the keyboard. Keyboard consists of a set of typewriter like keys that enable you to enter data into a computer. They have alphabetic keys to enter letters, numeric keys to enter numbers, punctuation keys to enter comma, period, semicolon, etc., and special keys to perform some specific functions. The keyboard detects the key pressed and generates the corresponding ASCII codes which can be recognized by the computer.

Figure 5.1 Keyboard

(a) .**<u>Disk Drives:</u>** (*storage*) is a device that reads or writes, or both on a disk medium. The disk medium may be either magnetic as with floppy disks or hard drives; optical as with CD-ROM (compact disc-read only memory) disks; or a combination of the two, as with magneto-optical disks.

(b) **<u>Monitor:</u>** (*output*) is a device connected to a computer that displays information on a screen (like a TV). Modern computer monitors can display a wide variety of information, including text, icons (pictures representing commands), photographs, computer rendered graphics, video and animation.

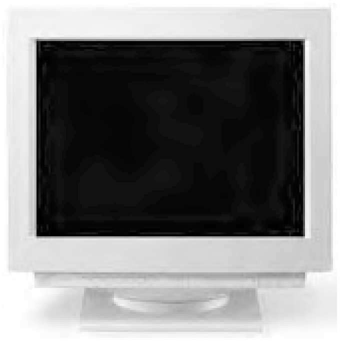


Figure 5.2 Display Monitor

(c) **<u>Mouse:</u>** (*input*) Mouse is an input device that controls the movement of the cursor on the display screen. Mouse is a small device, you can roll along a flat surface. In a mouse, a small ball is kept inside and touches the pad through a hole at the bottom of the mouse. When the mouse is moved, the ball rolls. This movement of the ball is converted into signals and sent to the computer. You will need to click the button at the top of the mouse to select an option. Mouse pad is a pad over which you can move a mouse. Mouse is very popular in modern computers.

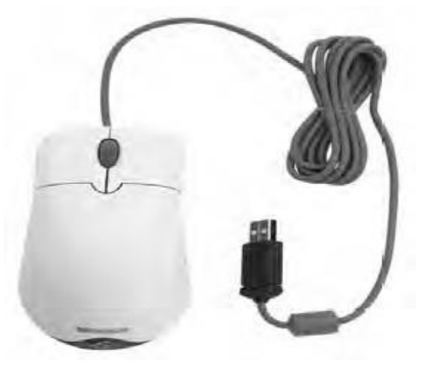


Figure 5.3 Mouse

(d) **<u>Modem:</u>** (*input/output*) the ordinary telephone lines transmit data in analog form. Computers are digital devices and use digital signals for data processing. Modems are used to connect digital computers with telephone lines. Modem at the originating computer modulates the digital signals and at the receiving computer demodulates analog signals. It converts digital signals into analog signals for transmission over telephone lines. At the other end of the channel, the modem converts the analog signals back into digital signals.

(e) **Printer:** (*output*) the printer takes the information on your screen and transfers it to a paper or hard copy. There are many different types of printers with various level of quality. The three basic types of printer are dot matrix, inkjet and laser.

Dot Matrix Printer (DMP) DMP prints one character at a time as a set of dots produced by the pins on the print head. It uses a nine or twenty-four pin print head. The pins or printing wires are aligned into the shape of the character to be printed before the print head strikes the ribbon. The impact of the strike produces character shapes on paper. The speed of DMP is measured in character per second (CPS).

Figure 5.4 Dot matrix

Ink jet Inkjet is a non-impact printer and is quiet when working. It sprays ink particles through its nozzle. On leaving the nozzle, the tiny particles of ink get electrically charged. The electrically charged particles are then guided on to the paper to form appropriate characters. Inkjet printers are as cheap as dot matrix printers are; but their operating costs are far higher than those of dot matrix printers. However, they give much better quality than DMPs. They are available in black and white and colour. The popular brands of inkjet printers are Hewlett-Packard, Epson Stylus and Canon.

Figure 5.5 Ink jet

Laser printer is a high-end printer. It is more expensive than inkjet printers and its operating costs are also higher than inkjet printers. It uses the same technology as that of Xerox copier machines and it can produce both character and graphic output. It gives the best quality output. Though expensive, laser printer is becoming increasingly popular.

Figure 5.6 Laser Printer

5.2 Software

Software, on the other hand, is the set of instructions a computer uses to manipulate data, such as word-processing program or a video game. These programs are usually stored and transferred via the computer's hardware to and from the CPU. Software also governs how the hardware is utilized: for example, how information is retrieved from a storage device. The interaction between the input and output hardware is controlled by the ***B**asic **I**nput **O**utput **S**ystem* (*BIOS*) software.

Software as a whole can be divided into a number of categories based on the types of work done by the programs. The two primary software categories are ***O**perating **S**ystems* (System Software), which control the workings of the computer, and application software, which addresses the multitude of tasks for which people use computers.

5.2.1 Systems Software

Are the programs that are dedicated to managing the computer itself, such as Operating System, file management utilities and disk operating system (DOS) The Operating System (*OS*) is a collection of programs that controls the application software that users run and provides a link between the hardware and software currently running on the computer. The OS has three major functions:

- It coordinates and manipulates computer hardware, such as computer memory, printer, disks, keyboard, mouse, and monitor;
- It organizes files on a variety of storage media such as floppy disks, hard drives, compact disks, digital video discs and tape;
- It manages hardware errors and loss of data.

5.2.2 Application Software

Application Software is designed to help the user perform one or more related specific tasks. Depending on the work for which it was designed, an application can manipulate text, numbers, graphics, or a combination of these elements. Application software thus include word processors, spreadsheets, database management, and many other *applications*.

CHAPTER SIX

ALGORITHMS AND FLOWCHARTS

6.1 Algorithm

The term **algorithm** originally referred to any computation performed via a set of rules applied to numbers written in decimal form. The word is derived from the phonetic pronunciation of the last name of *Abu Ja'far Mohammed ibn Musa al-Khowarizmi*, who was an Arabic mathematician who invented a set of rules for performing the four basic arithmetic operations (addition, subtraction, multiplication and division) on decimal numbers.

An algorithm is a representation of a solution to a problem. If a problem can be defined as a difference between a desired situation and the current situation in which one is, then a problem solution is a procedure, or method, for transforming the current situation to the desired one. We solve many such trivial problems every day without even thinking about it, for example making breakfast, travelling to the workplace etc. But the solution to such problems requires little intellectual effort and is relatively unimportant. However, the solution of a more interesting problem of more importance usually involves stating the problem in an understandable form and communicating the solution to others. In the case where a computer is part of the means of solving the problem, a procedure, explicitly stating the steps leading to the solution, must be transmitted to the computer. This concept of problem solution and communication makes the study of algorithms important to computer science. Throughout history, man has thought of ever more elegant ways of reducing the amount of labour needed to do things. A computer has immense potential for saving time/energy, as most (computational) tasks that are repetitive or can be generalised can be done by a computer. For a computer to perform a desired task, a method for carrying out some sequence of events, resulting in accomplishing the task, must somehow be described to the computer. The algorithm can be described on many levels because the algorithm is just the procedure of steps to take and get the result. The language used to describe an algorithm to other people will be quite different from that which is used by the computer; however the actual algorithm will in essence be the same.

Definition:

*An **algorithm** is procedure consisting of a finite set of unambiguous rules (instructions) which specify a finite sequence of operations that provides the solution to a problem, or to a specific class of problems for any allowable set of input quantities (if there are inputs). In other word, an **algorithm** is a step-by-step procedure to solve a given problem*

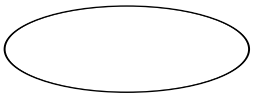
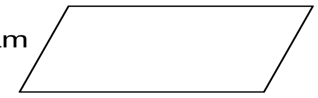
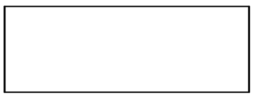
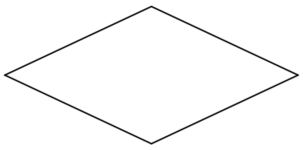
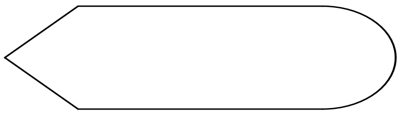
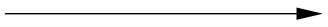
6.1.1 Pseudo Code

Pseudo code is one of the tools that can be used to write a preliminary plan that can be developed into a computer program. **Pseudo code** is a generic way of describing an algorithm without use of any specific programming language syntax. It is, as the name suggests, *pseudo* code —it cannot be executed on a real computer, but it models and resembles real programming code, and is written at roughly the same level of detail.
Pseudo code, by nature, exists in various forms, although most borrow syntax from popular programming languages (like **C**, **Lisp**, or **FORTRAN**). Natural language is used whenever details are unimportant or distracting.

6.2 Flowcharting

Flowcharting is a tool developed in the computer industry, for showing the steps involved in a process. A flowchart is a diagram made up of *boxes, diamonds* and other shapes, *connected by arrows* - each shape represents a step in the process, and the arrows show the order in which they occur. Flowcharting combines symbols and flow lines, to show figuratively the operation of an algorithm.
In computing, there are dozens of different symbols used in flowcharting (there are even national and international flowcharting symbol standards). In business process analysis, a couple of symbols are sufficient. A box with text inside indicates a step in the process, while a diamond with text represents a decision point.

6.2.1 Flowchart Symbols

Name	**Symbol**	**Use in Flowchart**
Oval		Denotes the beginning or end of the program
Parallelogram		Denotes an input operation
Rectangle		Denotes a process to be carried out e.g. addition, subtraction, division etc.
Diamond		Denotes a decision (or branch) to be made. The program should continue along one of two routes. (e.g. IF/THEN/ELSE)
Hybrid		Denotes an output operation
Flow line		Denotes the direction of logic flow in the program

6.3 Pseudo code, Flowchart, and Algorithm: Examples

Example 1

Write an algorithm to determine a student's final grade and indicate whether it is passing or failing. The final grade is calculated as the average of four marks.

Pseudo code:

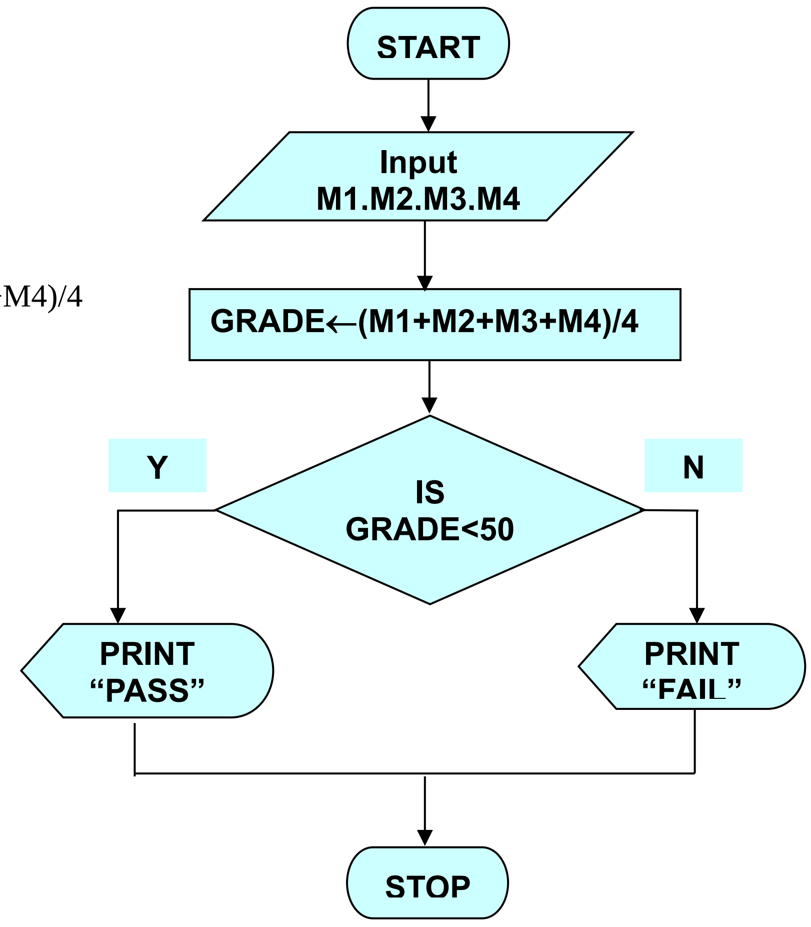
```
o   Input a set of 4 marks
o   Calculate their average by summing and dividing by 4
o   if average is below 50
            Print "FAIL"
```

```
        *else*
                *Print "PASS"*
```

Detailed Algorithm

```
Step 1: Input M1,M2,M3,M4
Step 2: GRADE ← (M1+M2+M3+M4)/4
Step 3: if (GRADE < 50) then
                Print "FAIL"
        else
                Print "PASS"
        endif
```

Flowchart



Example 2

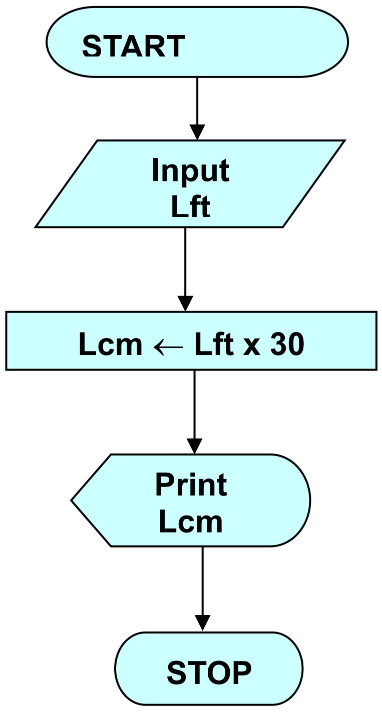
Write an algorithm and draw a flowchart to convert the length in feet to centimeter

Pseudo code:

- *Input the length in feet (Lft)*
- *Calculate the length in cm (Lcm) by multiplying LFT with 30*
- *Print length in cm (LCM)*

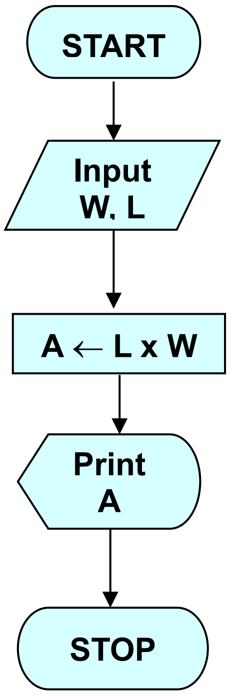
Algorithm

- Step 1: Input Lft
- Step 2: Lcm ← Lft x 30
- Step 3: Print Lcm



Example 3

Write an algorithm and draw a flowchart that will read the two sides of a rectangle and calculate its area.



Pseudo code

- *Input the width (W) and Length (L) of a rectangle*
- *Calculate the area (A) by multiplying L with W*
- *Print A*

Algorithm

- Step 1: Input W,L
- Step 2: A ← L x W
- Step 3: Print A

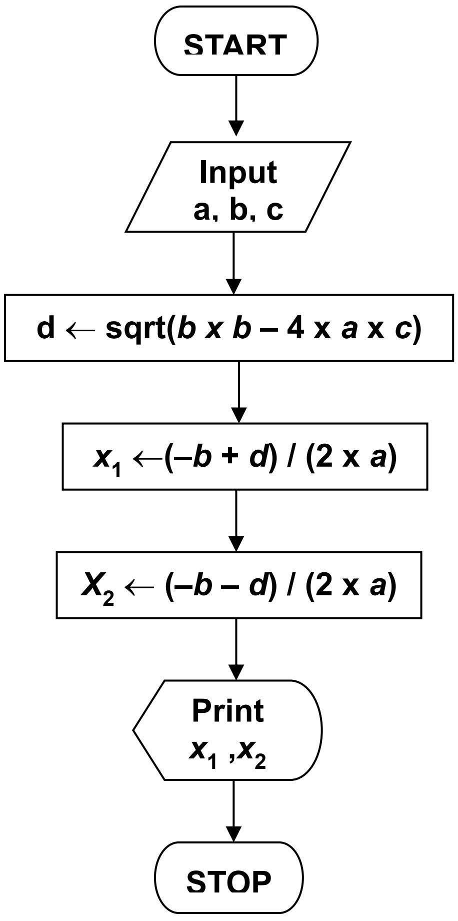
Example 4

Write an algorithm and draw a flowchart that will calculate

the roots of a quadratic equation

Hint: **d** = sqrt ($b^2 - 4ac$),

roots are: $\mathbf{x1} = (-b + d)/2a$ and $\mathbf{x2} = (-b - d)/2a$



Pseudo code:

- *Input the coefficients (a, b, c) of the quadratic equation*
- *Calculate* **d**
- *Calculate* **x1**
- *Calculate* **x2**
- *Print* x1 *and* x2

Algorithm:

- Step 1: Input a, b, c
- Step 2: $d \leftarrow$ sqrt ($b^2 - 4ac$)
- Step 3: $x1 \leftarrow (-b + d) / (2 \text{ x } a)$
- Step 4: $x2 \leftarrow (-b - d) / (2 \text{ x } a)$
- Step 5: Print $x1$, $x2$

6.4 Control Structures/Logical Structures

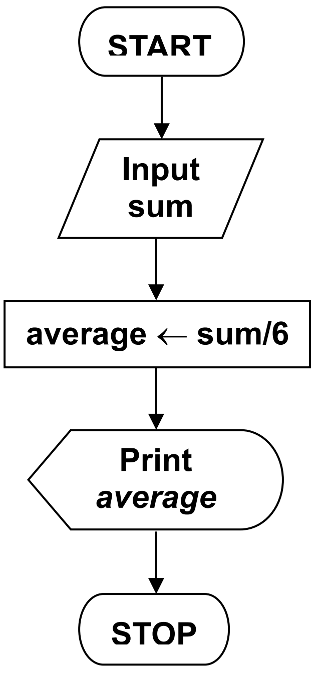
The key to better algorithm design and thus to programming lies in limiting the control structure to only three constructs. These are explained in the sections that follow.

6.4.1 The sequence structure

The first type of control structures is called the sequence structure. This structure is the most elementary structure. The sequence structure is a case where the steps in an algorithm are constructed in such a way that, no condition step is required. The sequence structure is the logical equivalent of a straight line. For example, suppose you are required to design an algorithm for finding the average of six numbers, and the sum of the numbers is given. The pseudo code will be as follows

- *Get the sum*
- *Average = sum / 6*
- *Output the average*

The corresponding flowchart will appear as follows:



6.4.2 Decision Structure or Selection Structure

If–Then–Else Structure

The structure is as follows

```
If (condition) then
    true alternative
else
    false alternative
endif
```

Decision Structure

The decision structure or mostly commonly known as a selection structure, is case where in the algorithm, one has to make a choice of two alternatives by making decision depending on a given condition. Selection structures are also called **case** selection structures when there are two or more alternatives to choose from.

This structure can be illustrated in a flowchart as follows:

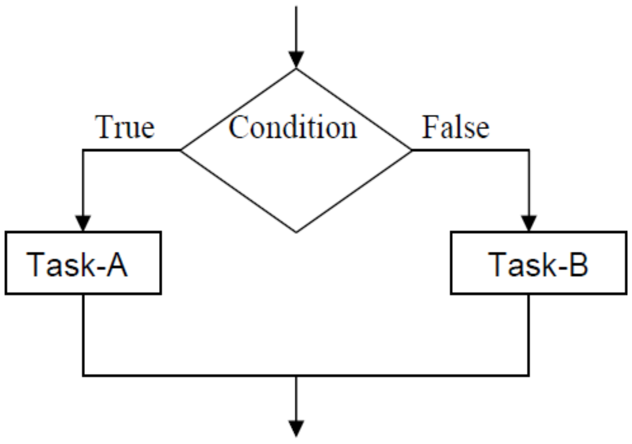


Figure 6.1 Decision Structure

In figure 6.1, the condition is evaluated, if the condition is true Task-A is evaluated and if it is false, then Task-B is executed. A variation of the construct of the above figure is shown below where if the condition is untrue, nothing is done, and Task-A is perform otherwise.

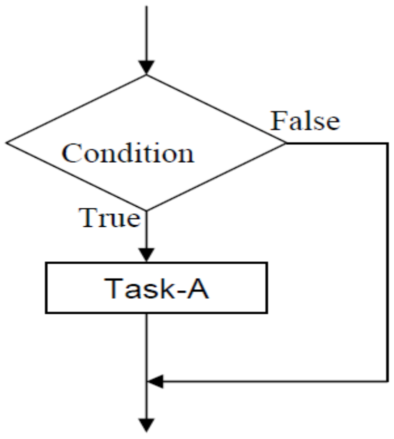


Figure 6.2 Alternative Scheme for Selection/Decision Structure

Relational operators are used in conditional statements to compare values to know the course of action. These operators are listed in the table below.

Table 6.1 List of Relation Operators

Relational Operators	
Operator	**Description**
>	**Greater than**
<	**Less than**

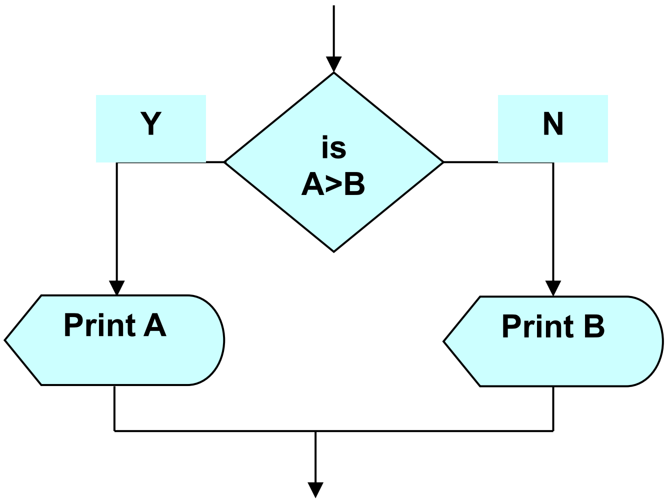
==	**Equal to**
≥	**Greater than or equal to**
≤	**Less than or equal to**
!=	**Not equal to**

The expression A>B is a logical expression. It describes a **condition** we want to test

- ***if A>B is true (if A is greater than B)*** *we take the action on left*
- print the value of A
- ***if A>B is false (if A is not greater than B)*** *we take the action on right*
- print the value of B

In pseudo code form we get

```
If condition is true
        Then print A
Else
        Print B
```

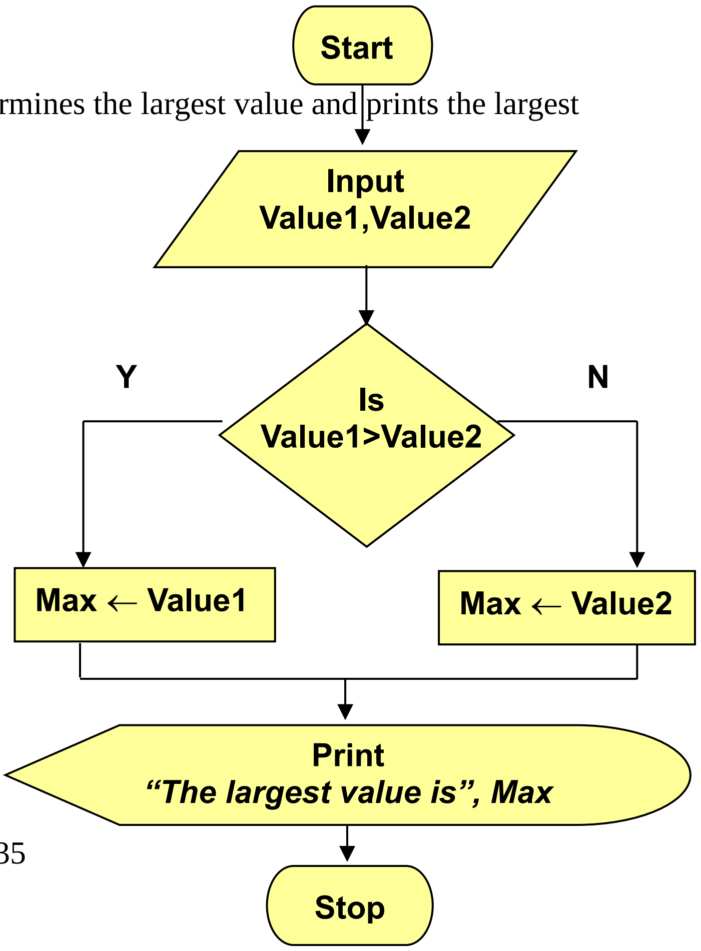


Example 5

Write an algorithm that reads two values, determines the largest value and prints the largest value with an identifying message.

Algorithm

```
Step 1:  Input Value1, Value2
Step 2: if (Value1 > Valu2) then
                Max ← Value1
        else
                Max ← Value2
        endif
Step 3: Print "The largest value is", Max
```



Nested Ifs

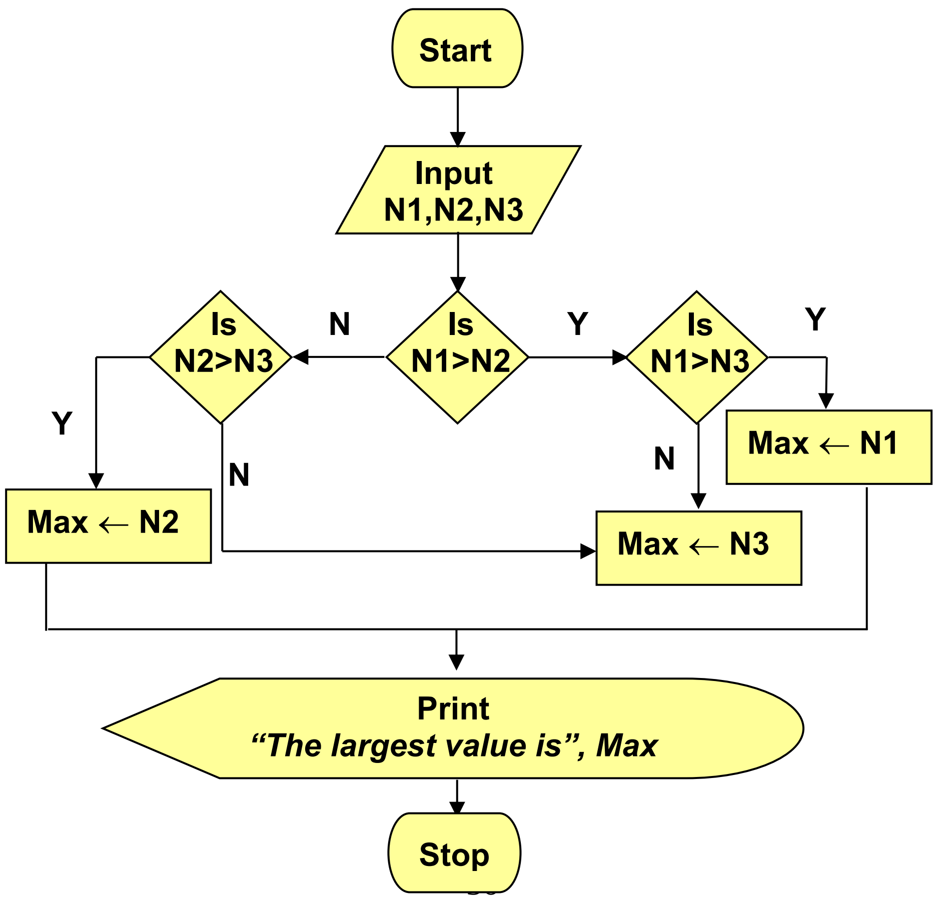
One of the alternatives within an IF–THEN–ELSE statement may involve further IF–THEN–ELSE statement. This is possible to allow for further test within a given alternative. We commonly use this type of nesting both in algorithmic and real programming constructs.

Example 6

Write an algorithm that reads **three** numbers and prints the value of the largest number

Algorithm

```
Step 1:  Input N1, N2, N3
Step 2: if (N1>N2) then
                if (N1>N3) then
                        MAX ← N1   [N1>N2, N1>N3]
                else
                        MAX ← N3   [N3>N1>N2]
                endif
        else
                if (N2>N3) then
                        MAX ← N2   [N2>N1, N2>N3]
                else
                        MAX ← N3   [N3>N2>N1]
                endif
        endif
Step 3: Print "The largest number is", MAX
```



Example 7

Write and algorithm and draw a flowchart to

a) read an employee name (NAME), overtime hours worked (OVERTIME), hours absent (ABSENT) and

b) determine the bonus payment (BONUS).

c) The BONUS is calculated as per table below

Bonus Schedule	
OVERTIME – (2/3)*ABSENT	Bonus Paid
>40 hours >30 but ≤ 40 hours >20 but ≤ 30 hours >10 but ≤ 20 hours ≤ 10 hours	N50 N40 N30 N20 N10

Algorithm

```
Step 1:  Input NAME,OVERTIME,ABSENT
Step 2: BONUS  ← OVERTIME – (2/3)*ABSENT
Step 2: if (BONUS > 40) then
               PAYMENT  ← 50
           else if (BONUS > 30) then
               PAYMENT  ← 40
           else if (BONUS > 20) then
               PAYMENT  ← 30
           else if (BONUS > 10) then
               PAYMENT  ← 20
           else
              PAYMENT ← 10
           endif
Step 3: Print "Bonus for", NAME "is N", PAYMENT
```

Exercise 1: Draw the flowchart of the above algorithm

Exercise 2: A man has to get a fox, a chicken, and a sack of corn across a river. He has a rowboat, and it can only carry him and one other thing. If the fox and the chicken are left together, the fox will eat the chicken. If the chicken and the corn are left together, the chicken will eat the corn. How does the man do it? Solve the problem and make a flow chart with solution.

Exercise 3:

CHAPTER SEVEN

PROGRAMMING AND PROBLEM SOLVING

7.1 What is Programming?

Computers do what we *tell* them to do, NOT what we *want* them to do. **Computer Programming** involves writing instructions and giving them to the computer to complete a task.

7.2 What is a Computer Program?

- A computer program is a set of instructions written in a computer language, in order to be executed to perform a specific task.
- There are hundreds of programming languages in use nowadays.
- Computer Programs are also named as <u>**SOFTWARE**</u>.
- There are different types of software available. Some of them are:
    - Operating System Software (Win 9x, 2000, Windows 7, 8, 10 Unix, Linux, Ubuntu …).
    - Compilers and Interpreters (Used with Programming Languages.).
    - Application Software (Payroll system, Accounting System).
    - Embedded System Software (Used in TV sets, telephones).
    - Utilities and Tools for productivity (MS Word, Excel, …).

7.3 Who is a Programmer?

A programmer is a person who writes the required computer programs. Programmers translate the expected tasks given in human understandable form into machine understandable form by using compilers and interpreters.

7.4 Compiler & Interpreter

<u>**Compiler**</u> is a specific software that gets the whole <u>***Source Code***</u> (Computer program written in human understandable form) and translates it into <u>***Object Code***</u> (Computer Program that is in machine understandable form) all at a time.

<u>**Interpreter**</u> is translating and executing one statement (command of Computer Program) of <u>***Source Code***</u> into <u>***Object Code***</u> at a time. (It means, interpreters translate and execute computer programs line by line).

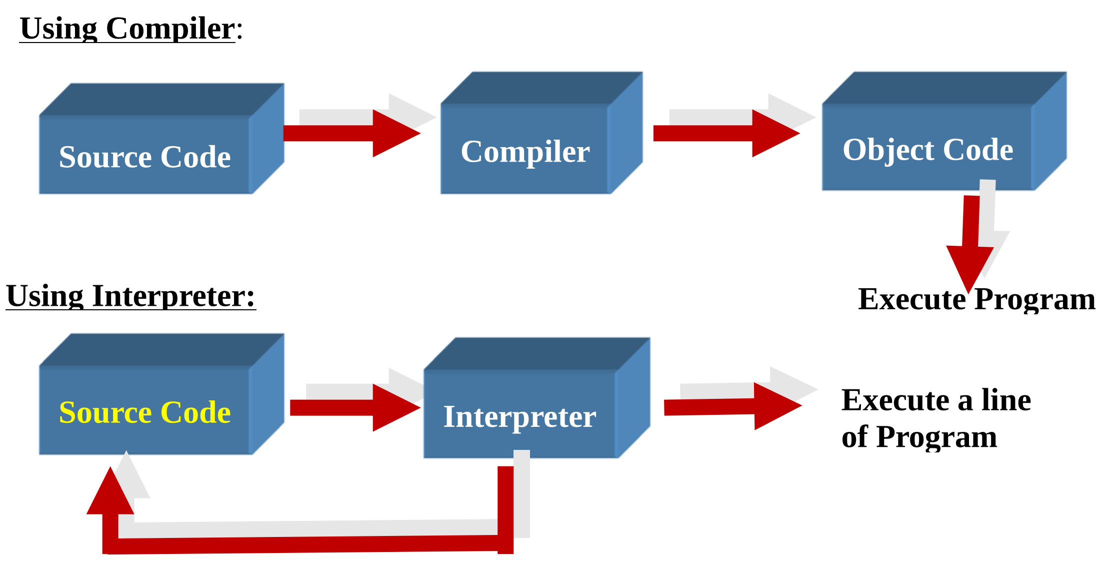


Figure 7.1 How Compiler and Interpreter Work

7.5 Features of Well-Designed Programs

Well-designed programs must be:

- Correct and accurate
- Easy to understand
- Easy to maintain and update
- Efficient
- Reliable
- Flexible

7.6 Programming Process

1. *Problem definition*

    - What must the program do?
    - What outputs are required and in what form?
    - What inputs are available and in what form?

Example: *Find a maximum of two numbers*

- Input two numbers, compare them and print the maximum value
- Inputs and outputs are decimal numbers
- Inputs are entered from the keyboard
- Result is shown on the monitor

2. *Program Design* involves creating an **algorithm** – sequence of steps, by which a computer can produce the required outputs from the available inputs

3. *Program Coding* means expressing the algorithm developed for solving a problem, in a programming language.
    - *Program Compilation* – translation of a program written in a high-level programming language into machine language instructions
    - *Compilation step* converts a source program into an intermediate form, called *object code*
    - *Linking step* is necessary to combine this object code with other code to produce an *executable program*
    - The advantage of this two-step approach:
        - Source of the large program may be split into more than one file
        - These files are worked on and compiled separately
        - Object code may be stored in libraries and used for many programs
        - Then they will be combined into one executable code
4. *Program Testing & Debugging*
    - Initially, almost all programs may contain a few errors, or *bugs*
    - *Testing* is necessary to find out if the program produces a correct result. Usually it is performed with sample data
    - *Debugging* is the process of locating and removing errors

7.7 Types of Errors

There are basically three types of errors as explained in the following sections.

7.7.1 Syntax Errors

Violation of syntactic rules in a Programming Language generates syntax errors. ***Effect***: Interpreter or Compiler finds it in Syntax Check Phase.

7.7.2 Semantic/Logical Errors

Doing logical mistakes causes semantic errors in Source code. ***Effect:*** Interpreters and Compilers cannot notice them, but on execution, they cause unexpected results.

7.7.3 Run-time Errors

Occur on program execution. Mostly caused by invalid data entry or tries to use non existing resources. ***Effect*** It occurs on run time and may crash the program execution

7.8 Types of Programming Languages

1. **Low-level languages** (Machine Language, Assembly Language).
    - This is also called ***Machine language.*** This is made up of binary 0s and 1s. This is the only language the computers can understand
        - ☐ ***advantages*** of machine languages are fast execution speed and efficient use of main memory

- ☐ ***disadvantages*** of machine languages are writing machine language is tedious, difficult and time consuming

2. **Assembly language** is a low-level interface to CPU functions

   - ■ Example:

```
Mov  ax, 1
Mov  bx, 2
Add     bx
```

   - ■ Writing programs can be very time-consuming, as you must directly manipulate CPU registers and use complicated interfaces to I/O devices
   - ■ Code is *assembled* to make Machine Language (consisting only of 1’s and 0’s) which is the real language of the CPU

3. **High Level Languages.**
   Slower, needs more resources but more user friendly (needs less programming time and it is easier to develop a program).
   - ■ **Examples of *high-level languages*** are *FORTRAN, COBOL, PL/I, BASIC, PASCAL, C, LISP, Prolog, Logo, C++, C, Java, Python*

CHAPTER EIGHT

INTRODUCTION TO PROGRAMMING WITH C++

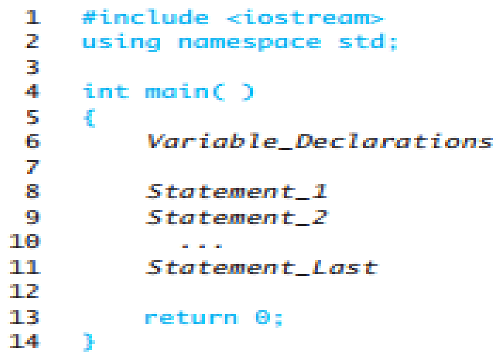
8.1 Brief History of C++

ANSI/ISO Standard C++

The programming language C++ evolved from C and was designed by Bjarne Stroustrup at Bell Laboratories in the early 1980s. From the early 1980s through the early 1990s, several C++ compilers were available. Even though the fundamental features of C++ in all compilers were mostly the same, the C++ language, referred to in this book as Standard C++, was evolving in slightly different ways in different compilers. As a consequence, C++ programs were not always portable from one compiler to another. To address this problem, in the early 1990s, a joint committee of the American National Standard Institution (ANSI) and International Standard Organization (ISO) was established to standardize the syntax of C++. In mid-1998, ANSI/ISO C++ language standards were approved. Most of today's compilers comply with these new standards. Over the last several years, the C++ committee met several times to further standardize the syntax of C++. In mid-2010, the second standard of C++ was voted on and approved. The main objective of this standard, referred to as C++ 0X, or tentatively as C++ 11, is to make the C++ code cleaner and more effective. For example, the new standard introduces the data type long to deal with large integers, auto declaration of variables using initialization statements, enhancing the functionality of for-loops to effectively work with arrays and containers, and new algorithms. However, not all new features of this new standard have been implemented by all the compilers currently available. In this book, we introduce the new C++ features that we know have been implemented by the well-known compilers and also comment on the ones that will be implemented in the future.

8.2 Layout of a Simple C++ Program

The general form of a simple C++ program is shown in Display below.



```
 1   #include <iostream>
 2   using namespace std;
 3
 4   int main( )
 5   {
 6       Variable_Declarations
 7
 8       Statement_1
 9       Statement_2
10         ...
11       Statement_Last
12
13       return 0;
14   }
```

Line 1: “**#include <iostream>**” is called an include directive. It tells the compiler where to find information about certain items that are used in your program. In this case *iostream* is the name of a library that contains the definitions of the routines that handle input from the keyboard and output to the screen.

Line 2: “using namespace std;” This line says that the names defined in iostream are to be interpreted in a “standard way” (std is an abbreviation of standard).

Line 4: “**int main( )”:** This line is part of every C++ program. It is called a ***main function***. C++ start program execution from the *main function*.

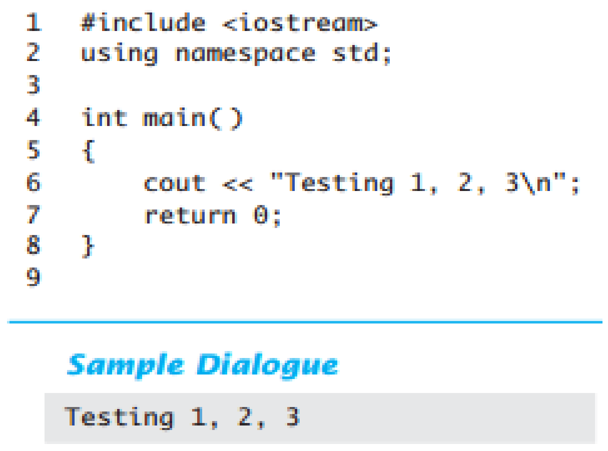
Line 5 and Line 14: The braces **{**and**}** mark the beginning and end of the main part of the program.

Line 6: Variable declarations is on this line. It is not necessary that all your variable declarations must be at the beginning of your program, but that is a good default location for them.

Line 8 to 11: The statements are the instructions that are followed by the computer. Statements are often called executable statements. A statement ends with a semicolon. The semicolon in statements is used in more or less the same way that the period is used in English sentences; it marks the end of a statement.

Line 13: “return 0;” says to “end the program when you get to here.” This line is called a return statement and is an executable statement because it tells the computer to do something; specifically, it tells the computer to end the program. The number 0 has no intuitive significance to us yet but must be there; its meaning will become clear as you learn more about C++.

8.3 Printing Line of Text on the Computer Screen



```
#include <iostream>
using namespace std;

int main( )
{
    cout << "Testing 1, 2, 3\n";
    return 0;
}

```

Sample Dialogue

```
Testing 1, 2, 3
```

When you run or compiled the above program, it will display the output shown in the dialogue above "Testing 1, 2, 3".

8.3.1 The Output Operator

The symbol << is called the *output operator* in C++. (it is also called the *put operator* or the *stream operator*) It insert values into the output stream that is named on its left. We usually used `cout` output stream, which ordinarily refers to the computer screen. So, the statement

```
cout <<100
```

would display 100 on the screen.

An *operator* is something that performs an action on one or more objects. The output operator << performs the action of sending the value of the expression listed on its right to the output stream listed on its left. Since the direction of this action appears to be from right to left, the symbol << was chosen to represent it. It should remind you of an arrow pointing to the left.

The `cout` object is called a "stream" because output set to it flows like a stream. If several things are inserted into the `cout` stream, they fall in line, one after the other as they are dropped into the stream., like leaves falling from a tree into a natural stream of water. The values that are inserted into the `cout` stream are displayed on the screen in that order.

Another "Hello, World" Program

```
int main()
{ // prints "Hello,W orld!":
cout << "Hel" << "lo, Wo" << "rld!" << endl;
}
```

The output operator is used four times here, dropping the four objects `"Hel"`, `"lo,Wo"` , `"rld!"`, and `endl` into the output stream. The first three are strings that are concatenated together (i.e., strung end-to-end) to form the single string `"Hello,World!"` . The fourth object is the stream manipulator object `endl` (meaning "end of line"). It does the same as appending the endline character '\n' to the string itself: it sends the print cursor to the beginning of the next line. It also "flushes" the output buffer.

8.3.2 Variables and their Declaration

A variable is a symbol that represents a storage location in the computer's memory. The information that is stored in that location is called the value of the variable. One common way for a variable to obtain a value is by an assignment. This has the syntax

```
variable = expression;
```

First the expression is evaluated and then the resulting value is assigned to the variable. The equals sign "=" is the assignment operator in C++.

Example Using Integer Variables

In this example, the integer 44 is assigned to the variable m, and the value of the expression m + 33 is assigned to the variable n:

```
int main()
{// prints "m = 44 and n = 77":
     int m, n;
     m = 44; // assigns the value 44 to the variable m
     cout << "m = " << m;
     n = m + 33; // assigns the value 77 to the variable n
     cout << " and n = " << n << endl;
}
```

Note in this example that both m and n are declared on the same line. Any number of variables can be declared together this way if they have the same type.

Every variable in a C++ program must be declared before it is used. The syntax is

```
specifier type name initializer;
```

where specifier is an optional keyword such as `const`, type is one of the C++ data types such as int, name is the name of the variable, and initializer is an optional initialization clause such as `= 44`.

The purpose of a declaration is to introduce a name to the program; i.e., to explain to the compiler what the name means. The *type* tells the compiler what range of values the variable may have and what operations can be performed on the variable. The location of the declaration within the program determines the scope of the variable: the part of the program where the variable may be used. In general, the scope of a variable extends from its point of declaration to the end of the immediate block in which it is declared or which it controls.

8.3.4 The Input Operator

In C++, input is almost as simple as output. The input operator >> (also called the get operator or the extraction operator) works like the output operator <<.

Example Using the Input Operator

```
int main()
{//tests the input of integers, floats, and characters:
int m, n;
cout << "Enter two integers: ";
cin >> m >> n;
cout << "m = " << m << ", n = " << n << endl;
double x, y, z;
cout << "Enter three decimal numbers: ";
cin >> x >> y >> z;
cout << "x = " << x << ", y = " << y << ", z = " << z << endl;
char c1, c2, c3, c4;
cout << "Enter four characters: ";
cin >> c1 >> c2 >> c3 >> c4;
cout << "c1 = " << c1 << ", c2 = " << c2 << ", c3 = " << c3
<< ", c4 = " << c4 << endl;
}
```

8.3.5 Rules for Naming an Identifier

Only letters, digits and single underscore characters are valid identifiers.

Neither spaces nor punctuation marks or symbols can be part of an identifier.

Variable identifiers always have to begin with a letter or with an underscore character (_).

Variable identifiers cannot match any keyword of the C++ language nor your compiler's specific ones.

C++ language is a "case sensitive" language that means an identifier written in capital letters is not the same with another one with the same name but written in small letters.

keywords in C++:

asm, auto, bool, break, case, catch, char, class, const, const_cast, continue, default, delete, do, double, dynamic_cast, else, enum, explicit, export, extern, false, float, for, friend, goto, if, inline, int, long, mutable, namespace, new, operator, private, protected, public, register, reinterpret_cast, return, short, signed, sizeof, static, static_cast, struct, switch, template, this, throw, true, try, typedef, typeid, typename, union, unsigned, using, virtual, void, volatile, wchar_t, while

8.5 Data types

The memory in our computers is organized in bytes. A byte is the minimum amount of memory that we can manage in C++. When programming, we store the variables in our computer's memory, but the computer has to know what kind of data we want to store in them.

Next you have a summary of the basic fundamental data types in C++, as well as the range of values that can be represented with each one:

Type	Byte*	Description	Range
`char`	1	character or integer 8 bits in length	`signed:` -128 to 127 `unsigned:` 0 to 255
`short`	2	integer 16 bits length	`signed:` -32,768 to 32,767 `unsigned:` 0 to 65,535
`long`	4	integer 32 bits length	`signed:` -2,147,483,648 to 2,147,483,647 `unsigned:` 0 to 4,294,967,295
`int`	*	Integer. Its length traditionally depends on the length of the system's Word type, thus in MSDOS it is 16 bits long (2 bytes), whereas in 32 bit systems (like Windows 9x/2000/NT/XP) it is 32 bits long (4 bytes)	see `short`, `long`
`float`	4	floating-point number	$3.4\times10^{\pm38}$ (7 digits)
`double`	8	Double precision floating point number	$1.7\times10^{\pm308}$ (15 digits)
`long double`	8	Long double precision floating point number	$1.2\times10^{\pm4932}$ (19 digits)

* *Values of columns Bytes and Range may vary depending on system.*

8.6 Declaring Variables

All variables must be declared prior to their used. Here is the general form of a declaration:

```
type variable_name;
```

For example, to declare **x** to be a **float**, **y** to be an integer, and **ch** to be a character, you would write;

```
float x;
int y;
char ch;
```

8.7 Initializing Variables

A variable can be initialized by following its name with an equal sign and an initial value. For example, this declaration assigns **count** an initial value of 100:

```
int count = 100;
```

Implementation of the algorithms in chapter 6
Example 1

Write a program to determine a student's final grade and indicate whether it is passing or failing. The final grade is calculated as the average of four marks.

```
#include <iostream>

using namespace std;

int main()
{
    int M1, M2, M3, M4, grade;
    cout << "Enter the four marks: ";
    cin>>M1>>M2>>M3>>M4;

    grade = (M1+M2+M3+M4)/4;
    if (grade < 50)
        cout<<"FAIL";
    else
        cout<<"PASS";

    return 0;
}
```

Example 2

Write a C++ program that accept a length in feet and convert it to centimeter.

```
#include <iostream>
```

```
using namespace std;

int main()
{
    float Lft, Lcm;
    cout << "Enter the length in feet: " ;
    cin>>Lft;

    Lcm = Lft * 30;
    cout <<"length in cm (LCM) is "<< Lcm;

    return 0;
}
```

Example 3

Write a C++ program that read in the two sides of a rectangle (length and width) and calculate the area.

```
#include <iostream>
using namespace std;

int main()
{
    double L, W, Area;
    cout << "Enters the Length and Width: ";
    cin >> L >> W ;
    Area = L * W;

    cout<<"The Area is: "<<Area;

    return 0;
}
```

Example 4

Write a C++ program that read in the coefficients of a quadratic equation a, b,& c and calculate the roots of the equation

Hint: **d** = sqrt ($b^2 - 4ac$), roots are: ***x*1** = $(-b + d)/2a$ and ***x*2** = $(-b - d)/2a$

```
#include <iostream>
#include <cmath>
using namespace std;

int main()
{
    int a, b, c, x1, d, x2;
    cout << "Enter the coefficient of a: ";
    cin >>a;
    cout << "Enter the coefficient of b: ";
    cin >>b;
    cout << "Enter the coefficient of c: ";
    cin >>c;

    d = sqrt(b * b - 4 *a * c);
    x1 = (-b + d) / (2 * a);
    x2 = (-b - d) / (2 * a);
    cout << "The result of x1 is:" << x1 << endl;
    cout << "The result of x2 is:" << x2 << endl;

    return 0;
}
```

Example 5

Write a C++ program that reads in two values, determines the largest value and prints the largest value with an identifying message.

```
#include <iostream>
using namespace std;

int main()
{
     int num1, num2, largest;
     cout << "Enter the two numbers: ";

     cin>> num1 >> num2;
     if (num1 > num2)
        largest = num1;
      else largest = num2;
     cout << "The largest among the two numbers is: "<<
     largest;

    return 0;
}
```

Example 6
Write a C++ program that reads in **three** numbers and prints the value of the largest number.

```
#include <iostream>
using namespace std;

int main()
{
    int num1, num2, num3, largest;
    cout << "Enter the three numbers: ";

    cin>> num1 >> num2 >>num3;
    if (num1>num2)
```

```
        if(num1>num3)
        largest = num1;
      else
         largest = num3;

       else if(num2 > num3)
       largest = num2;
      else
        largest = num3;

      cout << "The largest among the three numbers is: "<<
largest;
    return 0;
}
```

Example 7

```
#include <iostream>
using namespace std;

int main()
{
    string NAME;
    int OVERTIME, ABSENT, PAYMENT;
    cout << "Enter Employee name: ";
    cin>>NAME;
    cout << "Enter Employee OVERTIME: ";
    cin>>OVERTIME;
    cout << "Enter Employee ABSENT: ";
    cin>>ABSENT;

    double BONUS = OVERTIME - (2/3) * ABSENT;
        if (BONUS > 40)
                 PAYMENT = 50;
            else if (BONUS > 30)
                 PAYMENT = 40;
            else if (BONUS > 20)
                 PAYMENT = 30;
            else if (BONUS > 10)
                 PAYMENT = 20;
            else
                 PAYMENT = 10;
     cout<< "Bonus for " << NAME << " is N"<< PAYMENT;

    return 0;
}
```

CHAPTER NINE

INTRODUCTION TO APPLICATION PACKAGES

REFERENCES

1. Chapters One and Two come from Chapters One through Three of the book *Computer Organization and Architecture Designing for Performance Eighth Edition* by Williams Stallings published by Prentice Hall 2010
2. C++ The Complete Reference 4th Edition by Herbert Schildt, published by McGraw-Hill 2003
3. Schaum's Outline of Theory and Problems of Programming with C++ 2nd Edition by John R. Hubbard, published by McGraw-Hill 2000