UNIT 16 INTRODUCTION TO COMPUTERS

Structure **Page No.**

16.1 Introduction 5
Objectives
16.2 Binary, Octal and Hexadecimal Number Systems 5
Conversion from One System to Another
Representation of Negative Integers
16.3 Working of Computers 10
Execution of Instruction
16.4 Evolution of Computers 17
16.5 Computer Software 19
System Software
Application Software
16.6 Categories of Languages 21
Machine Language
Assembly Language
High Level Languages
Fourth Generation Programming Languages
16.7 Summary 25
16.8 Solutions/Answers 25

16.1 INTRODUCTION

This Unit provides an introduction to computers. You will not be examined on the contents of this Unit. This gives a brief history of computers and gives you an idea of how they work. We advise you to browse through this Unit, skipping all the topics that you are familiar with. We strongly recommend that you read at least the second section of this Unit.

In Sec. 16.2, we will summarise binary, octal and hexadecimal systems. In Sec. 16.3, we will give an introduction to computers and describe how they work. In Sec. 16.4, we will discuss the evolution of computers. In Sec. 16.5, we will discuss what is a software and what are the different kinds of software. In Sec. 16.6, we will discuss computer languages.

Objectives

After studying this unit, you should be able to

- explain the von Neumann architecture of computers;
- explain the binary, octal and hexadecimal systems;
- convert numbers from one system to another system;
- explain evolution of computers;
- explain the various types of softwares available; and
- explain the various categories of computer languages, their strengths and weaknesses.

16.2 BINARY, OCTAL AND HEXADECIMAL NUMBER SYSTEMS

In our daily life we use the decimal system to represent numbers. Recall that, in this system, there are ten symbols available, $0, 1, 2, 3, \ldots, 9$ and each digit has a place value.

For example, in the number 21, 2 has a place value 20. We can also consider 21 as $2 \times 10^1 + 1 \times 10^0$. Similarly, we can consider 223 as $2 \times 10^2 + 2 \times 10^1 + 2 \times 10^0$. In other words, we assign a weight of 10^0 to the first digit(from the right), a weight of 10^1 to the second digit from the right and so on. This way, we can represent as large a number as we want using only 10 symbols!

In the case of computers things are different. Since computers are electronic instruments, only two states are available: Either there is a current flow or there isn't. If we use the presence of current flow to represent 1 and its absence to represent 0, we have two symbols. So, we have to use binary system in computers to represent data in the inner working of the computers. Also, system programmers use octal and hexadecimal systems. In octal system we use 8 symbols $0, 1, 2, 3, \ldots, 7$ to represent numbers. The weights of the digits are $1 = 8^0$, 8^1, $8^2 = 64$ etc. In hexadecimal system, we use 16 symbols, 0,1, 2, 3,...,9,A, B, C, D, E, F to represent the digits.

Please note that there is nothing sacrosanct about the decimal system. In fact, Mayans, who lived in South America centuries ago used base 20 system. Babylonians used base 60! They had 60 different symbols! You can see them in Fig. 1.

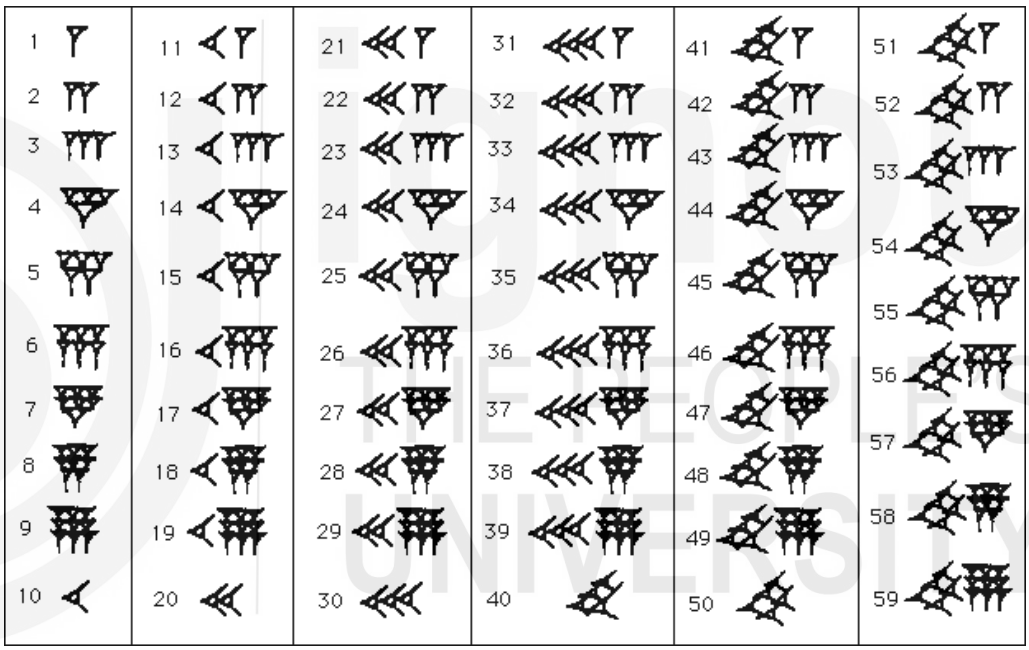


Fig. 1: Babylonian symbols.

We will discuss binary, octal and hexadecimal systems in this section. Since we will be representing numbers in various systems, we will specify the system by giving the base as subscript. For example 23_8 indicates representation is in base 8, 11101_2 indicates the representation is in base 2. If there is no subscript, the representation in in base 10. Let us now discuss how to convert from one system to another.

16.2.1 Conversion from One System to Another

As we mentioned before, we use two symbols, 0 and 1, in binary system. Here, each digit has a weight that is a power of 2. For example, the binary equivalent of 23 is 11101_2 in binary because
$1 + 1 \cdot 2 + 1 \cdot 2^2 + 1 \cdot 2^3 + 1 \cdot 2^4 = 1 + 1 \cdot 2 + 1 \cdot 4 + 0 \cdot 8 + 16 = 23$.

Let us now see how to convert from other systems to the decimal system. We start with conversion from binary to decimal, which is easy. Let us look at an example.

Example 1: Convert 110111_2 to decimal.

Solution: In decimal system, the right most digit is called the **least significant digit**

Table 1: Conversion from binary to decimal

Digit	Weight	Face value	Product
1	1	1	1
1	2	1	2
0	4	0	0
1	8	1	8
1	16	1	16
1	32	1	32
			59

and the left most digit is called the **most significant digit**. In binary representation, the right most digit is also called the **least significant bit(LSB)** and the right most digit is called the **most significant bit(MSB)**. Here is the step-by-step procedure for converting a number from binary to decimal:

1) In the first column of the first row, put the least significant bit. In the second column, put the corresponding weight, which is 1. In the third column, put the face value. (If you are puzzled why we have a column for the face value although it is the same as the digit, please be patient. You will understand the reason when we look at Example 2, where we show how to convert from hexadecimal to decimal.) Multiply the values in the second and third columns and put it in the fourth column.
2) Put the second digit from the right in the first column of the second row. In the second column, put the weight corresponding to the second digit. Note that, we can find the weight for a particular digit by multiplying the weight of the previous digit by the base. Here, base happens to be 2. So, multiplying the previous weight by 2, the weight is 2. In the third column, put the face value. Multiply the second and third columns and put the product in the fourth column. Carry this out for all the digits.
3) Add up all the numbers in the fourth column to get the decimal representation of the number.

* * *

Conversion from octal to decimal and hexadecimal to decimal are also similar as the following example shows.

Example 2:

a) Convert 2212_8 to decimal.

b) Convert $A314F_{16}$ to decimal.

Solution:

a) The procedure is similar to the one we used for the binary system. Note that, in octal system, the base is 8 and the weights are powers of 8.

Table 2: Conversion from octal to decimal

Digit	Weight	Face value	Product
2	1	2	2
1	8	1	8
2	64	2	128
2	512	2	1024
			1162

b) Here, the base is 16 and the weights are powers of 16. Note that, here, A, B, C, D, E, F are used to represent 11, 12, 13, 14 and 15, respectively. Here, the digit is not the same as face value!(See Table 3.)

Table 3: Conversion from hexadecimal to decimal

Digit	Weight	Face value	Product
F	1	15	15
4	16	4	64
1	256	1	256
3	4096	3	12288
A	65536	10	655360
			668283

* * *

Here are some exercises for you.

E1) Convert the following numbers to decimal system:
a) 111101_2 b) 334643_8 c) $A123BC_{16}$

Converting from decimal to other systems is also easy. Here is an example:

Example 3: Convert the following numbers from decimal to the system indicated:

a) Convert 59 to binary.

b) Convert 1162 to octal.

c) Convert 10560444 to hexadecimal.

Solution:

a) See Table 4a. We divide the number by 2. The remainder gives the least significant bit. We divide quotient by 2. This will give the next digit. We continue like this till we get 1 as the quotient. The last quotient and the successive remainders give the digits of the number in binary. The last quotient gives the most significant digit and the first remainder gives the least significant digit. We carry out this for the number 59. The calculation is given in Table 4a. So, 59 is 111011_2.

Table 4: Conversion from decimal to other systems

(a) Conversion to binary.

2	59	
2	29	$\rightarrow 1$
2	14	$\rightarrow 1$
2	7	$\rightarrow 0$
2	3	$\rightarrow 1$
	1	$\rightarrow 1$

(b) Conversion to octal.

8	1162	$\rightarrow 2$
8	145	$\rightarrow 1$
8	18	$\rightarrow 2$
	2	

(c) Conversion to Hexadecimal.

16	10560444	$\rightarrow 11(=\mathrm{C})$
16	660027	$\rightarrow 10(=\mathrm{B})$
16	42151	$\rightarrow 3$
16	2578	$\rightarrow 2$
16	161	$\rightarrow 1$
	$10(=\mathrm{A})$	

b) See Table 4b. In this case we divide by 8. As before, we repeatedly divide by 8 till we get a quotient less than 8. The successive remainders and the last quotient will give the digits of the number in octal. The last quotient which is less than 8 will give the most significant digit. So, 1162 is 2212_8.

c) See Table 4c. In this case, we divide by 16. Note that, the last quotient 10 is represented by the letter A. So, 1056044 is $A123BC_8$ in hexadecimal.

* * *

E2) Convert the following numbers from decimal to the system indicated:
   a) 61 to binary.
   b) 113059 to octal.
   c) 10560446 to hexadecimal.

It is also easy to convert from binary to octal or hexadecimal. Let us take an example. First, let us see how to convert from binary to octal. Suppose, we want to convert 11111101_2 into octal system. Starting from the right we group the digits in bunches of three:

$$\underbrace{11}\ \underbrace{111}\ \underbrace{101}$$

The last bunch has only two digits. Now, we find the value of each group in decimal. For example, the value for the first bunch from the right, 101, is 5. Similarly, we find the value of the remaining bunches.

$$\underbrace{11}_{3}\ \underbrace{111}_{7}\ \underbrace{101}_{5}$$

These values give the digits of 11111101_2 in octal, i.e. the value of the number in octal is 375_8.

We can convert a binary number to hexadecimal similarly. Here, we have to group the digits in bunches of 4. For example, suppose we want to convert the number 1111011110_2 to hexadecimal. We start from the right and group the digits in bunches of 4.

$$\underbrace{11}\ \underbrace{1101}\ \underbrace{1110}$$

As before, we find the value of each bunch.

$$\underbrace{11}_{3}\ \underbrace{1101}_{13}\ \underbrace{1110}_{14}$$

So, the number in hexadecimal is $3DE_{16}$.

E3) Convert 111110101_2 into octal and hexadecimal systems.

In all the examples we have discussed, all the integers were positive. In the next subsection, we will discuss how to represent negative numbers.

16.2.2 Representation of Negative Integers

As we mentioned earlier, at the lowest level, the CPU recognises only 0s and 1s. How can we store and work with negative numbers? The solution is to reserve the left most bit for the sign; it is 1 if the number is negative and 0 if it is positive.

While this solves the representation problem it still doesn't solve the problem of working with such numbers. Assuming that we are using 8 bits to store numbers for simplicity, we store 1 as $0000\ 0001_2$ and -2 as $1000\ 0001_2$. But, if we add them, we get $1000\ 0011_2$ which is -3 according to our notation!

One way to get round this problem is the 1's complement arithmetic. In this representation, we first represent the number without the sign and then complement

each bit; we replace 0 by 1 and 1 by 0. For example, 5 is $0000\ 0101_2$. If we complement the bits we get $1111\ 1010_2$ so -5 is $1111\ 1010_2$. If we add $2 = 0000\ 0010_2$ to $-5 = 1111\ 1010_2$, we get $1000\ 1100_2$ which is -3 because $3 = 0000\ 0011_2$ and if we complement the bits we get $1000\ 1100_2$. So, addition seems to work fine. But what happens when we add 2 and -2? We get $1111\ 1111_2$ which is '-0' in 1's complement. But, there is no such thing in Mathematics!

The way around this problem is to use 2's complement arithmetic. In this, we add 1 to the 1's complement representation. So, -2 is $1111\ 1101_2$ in 1's complement and $11111101_2 + 1 = 11111110_2$ in 2's complement. Now,

$$-2 + 2 = 1111\ 1110_2 + 0000\ 0010 = 0000\ 0000_2$$

with the last carry from got from adding the most significant bits of the numbers discarded. Now, there is only one zero and everything is fine! Here are some exercises for you to check your understanding of 2's complement arithmetic.

E4) Write -10 and -17 in 2's complement notation.

E5) Check that 2's complement representation works fine with multiplication also. Represent -3 and -5 in 2's complement notation and multiply them. Check that your answer is correct.

We end this section here. In the next section, we will introduce you to the working of computers.

16.3 WORKING OF COMPUTERS

Computer is defined in the Oxford dictionary as "An automatic electronic apparatus for making calculations or controlling operations that are expressible in numerical or logical terms."

The definition clearly categorises computer as an electronic apparatus although the first computers were mechanical and electro-mechanical apparatus. The definition also mentions the two major areas of computer application viz. data processing and computer assisted controls/operations. We can also infer from the definition that the computer can perform only Logical operations or Numerical operations.

The concept of computer is quite old. A hypothetical machine was defined in 1935–36 by Alan Turing, which was used for computability theory proofs. This is known as a **Turing machine**. It consists of an infinitely long 'tape' with symbols written at regular intervals. The tape may be infinite in one direction only, with the understanding that the machine will halt if it tries to move off the other end. All computer instruction sets, high level languages and computer architectures, including parallel processors, can be shown to be equivalent to a Turing Machine. Thus, all of them are equivalent to each other in the sense that if a problem can be solved by one of them, it can be solved by all the others. While serving on the BRL(Ballistic Research Laboratories) Scientific Advisory Committee, von Neumann joined the developers of ENIAC, the first tube computer and made some critical contributions. In 1947, while working on the design for the successor machine, EDVAC, von Neumann realised that ENIAC's lack of a centralised control unit could be overcome to obtain a rudimentary stored program computer. His ideas lead to what is now often called the **von Neumann architecture**. Most of today's computer designs are based on the concepts developed by John von Neumann, referred to as the von Neumann architecture. Von Neumann proposed that there should be a unit performing arithmetic and logical operation on the data. This unit

John von Neumann (1903-1957)

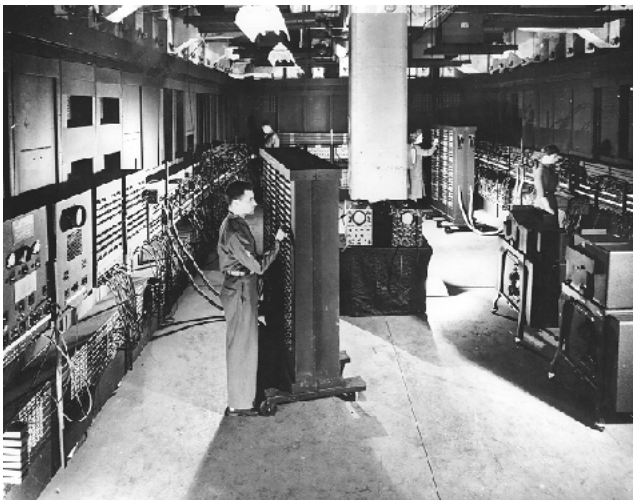


Fig. 2: ENIAC computer.

is termed as **Arithmetic Logic Unit (ALU)**. One of the ways to provide instruction to such computer will be by connecting various logic components in such a fashion that they produce the desired output for a given set of inputs. The process of connecting various logic components in specific configuration to achieve desired results is called **programming**. This programming, done by providing instructions within hardware by various connections, is termed as **hard-wired**. But this is a very inflexible process of programming.

Let us now look at the general configuration for arithmetic and logical functions. We need to have a control signal, which directs the ALU to perform a specific arithmetic or logic function on the data. By changing the control signal, we can perform the desired function on the data. We can perform any operation by providing appropriate signals. Thus, for a new operation it is enough to change control signals.

But, how can we supply these control signals? Let us try to answer this from the definition of a program. A program consists of a sequence of steps. Each of these steps requires certain arithmetic or logical or input/output operation to be performed on the data. Therefore, each step may require a new set of control signals. Is it possible for us to provide a unique code for each set of control signals? Well, the answer is 'yes'. But what do we do with these codes? It is necessary to have a hardware segment, which accepts a code and generates control signals. The unit, which interprets a code to generate respective control signal, is termed as **Control Unit (CU)**. Thus, a program consists of a sequence of codes. Now, this machine is quite flexible, as we only need to provide a new sequence of codes (program) for a new task to be performed. Each code is, in effect, an instruction, for the computer. The hardware interprets each of these instructions and generates respective control signals.

The Arithmetic Logic Unit (ALU) and the Control Unit (CU) together are termed as the **Central Processing Unit (CPU)**. The CPU is the most important component of a computer's hardware. The control unit interprets instructions and produces the respective control signals. The ALU performs the arithmetic operations such as addition, subtraction, multiplication and division, and the logical operations such as: 'Is A = B' (where A and B are both numeric or alphanumeric data) etc.

All the arithmetic and logical operations are performed in the CPU in special storage areas called **registers**. The size of the register is one of the important considerations in determining the processing capabilities of the CPU. Register size refers to the amount of information (number of bits) that can be held in a register at a time for processing. The larger the register size, the faster may be the speed of processing. A CPU's

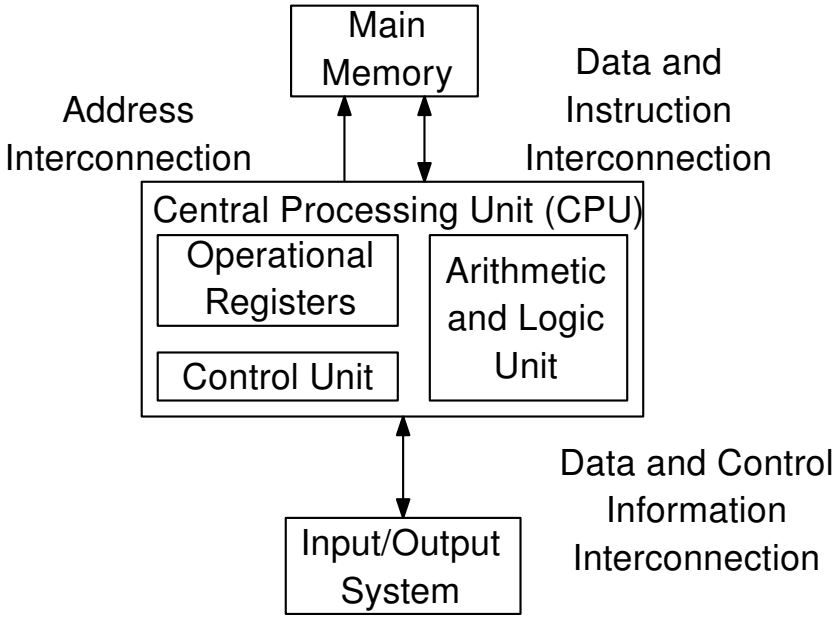


Fig. 3: Von Neumann architecture of a computer.

We will discuss the evolution of computers in Sec. 16.4.

processing power is measured in Million Instructions per Second (MIPS).The performance of the CPU was measured in milliseconds (one thousandth of a second) on the first-generation computers, in microseconds (one millionth of a second) on second-generation computers, in nanoseconds (one billionth of a second) on third-generation computers, and is measured in picoseconds (one 1000th of a nanosecond) or femtoseconds (one 1000th of a picosecond) in the later generations.

A von Neumann machine has only a single path between the main memory and control unit (CU). This constraint is referred to as **von Neumann bottleneck**. Several other architectures have been suggested for modern computers.

Now, how can we feed the instructions and data into computers? An external environment supplies the instruction and data, therefore, an **input module** is needed. The main responsibility of input module is to put the data in the form of signals that can be recognised by the system. Similarly, we need another component, which will report the results in proper format and form. This component is called **output module**. These components are referred together as **input/output (I/O) devices** or **peripheral devices**. Most common input/output devices are keyboard, mouse, monitor and printer. In addition, to transfer the information, the computer system internally needs the system interconnections called **Bus** which carries data and addresses.

Are these two components sufficient for a working computer? No, because input devices can bring instructions or data only sequentially and a program may not be executed sequentially as jump instructions are normally encountered in programming. In addition, more than one data element may be required at a time. Therefore, a temporary storage area is needed in a computer to store temporarily the instructions and the data. This component is referred to as **memory**. It was pointed out by von Neumann that the same memory can be used for storing data and instructions. In such cases the data can be treated as data on which processing can be performed, while instructions can be treated as data, which can be used for the generation of control signals.

The memory unit stores all the information in a group of memory cells, also called memory locations, as binary digits. Each memory location has a unique address and can be addressed independently. The contents of the desired memory locations are provided to the CPU by referring to the address of the memory location. The amount of information that can be held in the main memory is known as memory capacity. The

capacity of the main memory is measured in kilobytes (kB) or Megabytes (MB). One kilobyte equals 2^{10} bytes, which are 1024 bytes (or approximately 1,000 bytes). A megabyte equals 2^{20} bytes, which is approximately little over one million bytes (1,048,576 bytes to be precise). Since the memory on the computer chip is limited, in order to provide extended memory, devices like hard discs, floppy discs, compact discs (CD), magnetic tapes are used as external storage (memory) devices.

The basic function performed by a computer is the execution of a program. A program is a sequence of instructions, which operates on a data to perform certain tasks. In the computers data is represented in binary form by using two symbols 0 and 1, which are called binary digits or **bits**. But the data which we deal in practice consists of numeric data and characters such as decimal digits 0 to 9, alphabets A to Z, arithmetic operators (e.g. +,−, etc.), relations operators (e.g. =, > , etc.), and many other special characters (e.g.;@,{,], etc.). In general, computers use eight bits to represent a character internally. This allows up to 256 different items to be represented uniquely. This collection of eight bits is called **a byte**. Thus, one byte is used to represent one character internally. Most computers use two bytes or four bytes to represent numbers (positive and negative) internally. Another term, which is commonly used in computer, is a **word**. A word may be defined as a unit of information, which a computer can process, or transfer at a time. A word, generally, is equal to the number of bits transferred between the central processing unit and the main memory in a single step. It may also be defined as the basic unit of storage of integer data in a computer. Normally, a word may be equal to 8, 16, 32 or 64 bits. The terms like 32-bit computer, 64-bit computers, etc. basically refer to the word size of the computer.

The von Neumann machine uses stored program concept, i.e., the program and data are stored in the same memory unit. The computers prior to this idea used to store programs and data on separate memories. Entering and modifying these programs were very difficult as they were entered manually by setting switches and plugging and unplugging.

E6) State whether following statements are true or false.

i) A byte is equal to 8 bits and can represent a character internally.

ii) Von Neumann architecture specifies different memory for data and instructions. The memory, which stores data, is called data memory, which stores instructions, is called instruction memory.

iii) In Von Neumann architecture each bit of memory can be accessed independently.

iv) A program is a sequence of instructions designed for achieving a task/goal.

v) One MB is equal to 1024 kB.

In the next subsection, we discuss how the instructions in a program are executed in a computer.

16.3.1 Execution of Instruction

The main aspect in program execution is the execution of an instruction. The key questions, which can be asked in this respect, are: (a) how are the instructions supplied to the computer? and (b) how are they interpreted and executed? We will answer these questions now.

Execution of instructions in von Neumann machine is carried out in a sequential fashion (unless explicitly altered by the program itself) from one instruction to the next.

The program, which is to be executed, is a set of instructions, stored in the memory in form of **machine language** which comprises of '0's and '1's. The central processing unit (CPU) executes the instructions of the program to complete a task. The instruction execution takes place in the CPU registers.

Let us, first discuss few typical registers, which are generally available in the machines. These registers are:

Memory Address Register (MAR): It specifies the address of memory location from which data or instruction is to be accessed (for read operation) or to which the data is to be stored (for write operation).

Memory Buffer Register(MBR): This register contains the data to be written in the memory (for write operation) or it receives the data from the memory (for read operation).

Program Counter (PC): It keeps track of the instruction, which is to be executed next, after the execution of the on-going instruction.

Please do not confuse *personal computer* with *program counter* which is also often referred to as PC.

Instruction Register (IR): Here the instructions are loaded before their execution.

The simplest model of instruction processing can be a two-step process. The CPU reads ('fetches') instructions (codes) from the memory one at a time, and executes or performs the operation specified by this instruction. The fetch operation has to be carried out for all the instructions. It involves reading of an instruction from a memory location to the CPU. The execution of this instruction may involve several operations depending on the nature of the instruction.

The processing needed for a single instruction (fetch and execution) is referred to as instruction cycle. The instruction cycle consists of the fetch cycle and the execute cycle. Program execution terminates if the electric power supply is discontinued or some sort of unrecoverable error occurs; Also, a program can terminate itself.

For fetch cycle, a typical CPU uses a program counter (PC). PC keeps track of the instruction which is to be fetched next. Normally next instruction in sequence is fetched next as programs are executed in sequence.

The fetched instruction is in the form of binary code and is loaded into an instruction register (IR) in the CPU. The CPU interprets the instruction and does the required action. In general, these actions can be divided into the following categories:

Data Transfer: From the CPU to memory or from memory to CPU, or from CPU to I/O or I/O to CPU.

Data Processing: A logic or arithmetic operation performed by the CPU on the data.

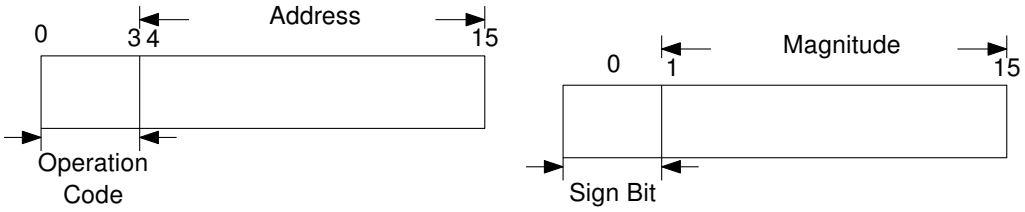
The data and address are usually expressed in hex code and indicated by a suffix H.

Sequence Control: This action may require alteration of sequence of execution. For example, an instruction from location 100 H on execution may specify that the next instruction should he fetched from location 200 H. On execution of such an instruction the Program Counter that was having location value 101 H (the next instruction to be fetched in case where memory word is equal to register size) will be modified to contain a location value 200 H.

Execution of an instruction may involve any combination of these actions. Let us understand the process with an example.

Example 4: Let us assume a hypothetical machine which has a 16 bit instructions and data. Each instruction of the machine consists of two components: (a) operation code (op-code) and (b) address of the operand in memory.

The op-code is assumed to be of 4 bits; therefore, remaining 12 bits are for the address of the operand as shown in Fig. 4a. The memory is divided into words of 16 bits. The CPU of this machine contains a register called Accumulator (AC) in addition to the registers given earlier. The AC stores the data temporarily. Fig. 4a shows the data format, which caters for sign bit. The machine can have $2^4 = 16$ possible operation



(a) Instruction Format. **(b) Data Format.**

Fig. 4: Instruction and data format for a typical machine.

codes. For example, let us assume operation with codes:

0001 as "Load the accumulator with the content of memory"
0010 as "Store the current value of Accumulator in the memory"
0011 as "Add the value from memory to the Accumulator"

This machine can address $2^{12} = 4096$ memory words directly. Remember that in this case the program counter is of 12 bits to accommodate address of the memory where the program resides.

Suppose we want to load the content at the memory location 760 H. 760 H is $0111\ 0110\ 0000_2$ in binary. The opcode for loading the memory content on to the Accumulator is 0001. So, the instruction is 0001 0111 0110 0000 in binary.

Note that the first four bits contain the opcode and the next 12 bits contain the address on which the operation is to be carried out. We can write the instruction succinctly as 1760 H in hexadecimal.

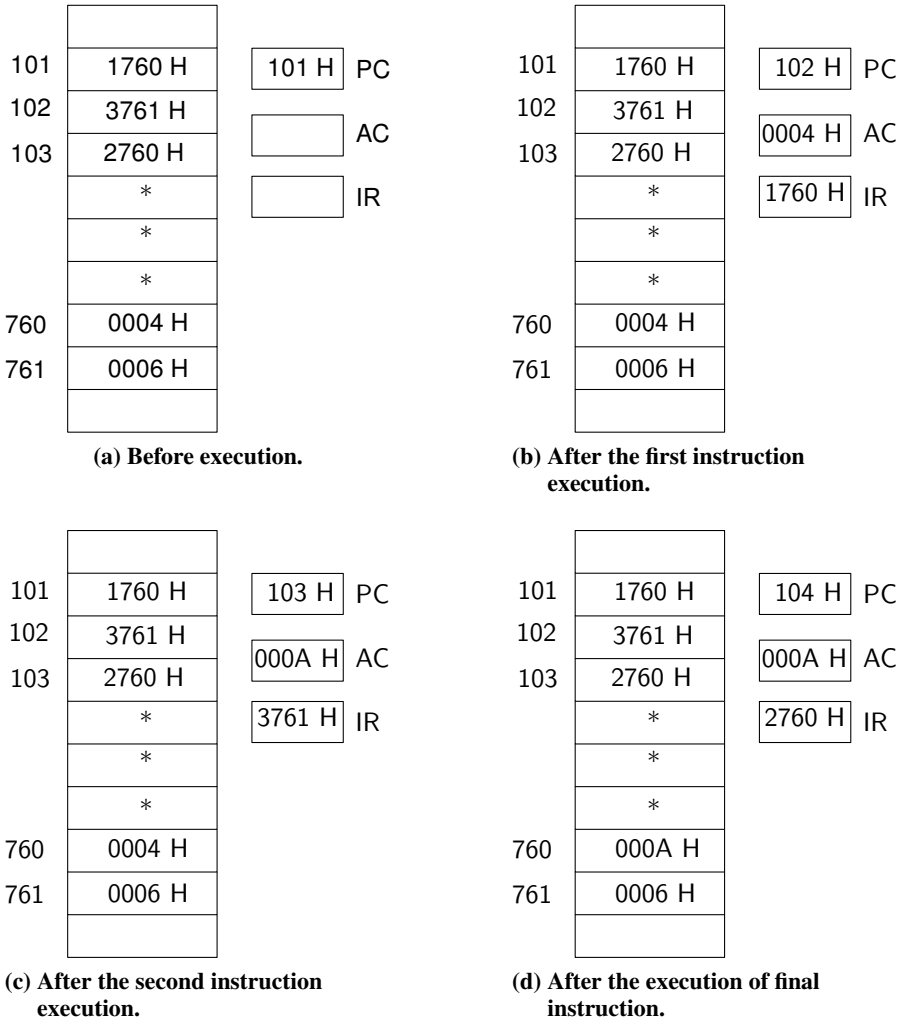
Let us assume that three consecutive instructions to be executed are

Instruction code	**Operation desired**
0001	0111 0110 0000 (Load accumulator with content at memory location 760 H)
0011	0111 0110 0001 (Add value from memory 761 H to accumulator)
0010	0111 0110 0000 (Store content of AC in the memory location 760 H)

The hexadecimal notation for these instructions is:

1760H
3761H
2760H

Let us assume that these instructions are stored in three consecutive memory locations 101 H, 102 H and 103 H and the PC contains a value (101 H), which is the address of first of these instructions as shown in Fig. 5a on the next page.) Since the PC contains value 101 H, so on the next instruction cycle the address stored in PC is passed to MAR which in turn helps in accessing the memory location 101 H and brings its content in MBR which in turn passes it on to IR. The PC is incremented to contain 102 H now. The IR has the value 1760 H that is decoded as "Load the content of address 760 H in the accumulator". (Refer to Fig. 5b on the following page.) Thus, the accumulator register is loaded with the content of location 760 H that is 0004 H.



(a) Before execution.

(b) After the first instruction execution.

(c) After the second instruction execution.

(d) After the execution of final instruction.

Fig. 5: Memory and register content during execution of instructions.

Now the instruction 101 H execution is complete, and the next instruction at 102 H (indicated by PC) is fetched and PC is incremented to 103 H. This instruction is 3761 H which implies "add the content of memory location 761 H to the accumulator". Therefore, accumulator will now contain the sum of its earlier value and the value stored in memory location 761 H. (Refer to Fig. 5c.)

On execution of the instruction at memory location 103 H, PC becomes 104 H; the accumulator results are stored in location 760 H and IR still contains the third instruction. This state is shown in Fig. 5d.

* * *

Please note that the execution of the instructions in the above example require only data transfer and data processing operations. All the instructions of the example required one memory reference during their execution. Does an instruction require more than one memory reference?

Well, yes! One such instruction is ADD A, B. The execution cycle of this instruction may consist of steps such as:

1. Decode the instruction, which is ADD.
2. Read memory location A (only the contents) into the CPU.
3. Read memory location B (only the contents) into the CPU. (Here, we are

assuring that the CPU has at least two registers for storing memory location contents. The contents of location A and location B needs to be written in two different registers.)

4. Add the values of the two above registers.

5. Write back the result from the register containing the sum (in the CPU) to the memory location B.

Thus, in general, the execution cycle for a particular instruction may involve more than one stage and memory references. In addition, an instruction may ask for an I/O operation. For such purpose the ports where the I/O devices are connected are allotted addresses just like any other memory location. When a data from such device is required, the address of the device is placed in the instruction instead of address of a memory location.

E7) What will be the steps involved in interchanging the data in memory location A and memory location B?

After understanding the fundamental operation of a computer, let us now briefly discuss the evolution of computers over the years.

16.4 EVOLUTION OF COMPUTERS

You know that, with the growth in microelectronics the IC(integrated circuit) technology evolved rapidly. One of the major milestones in this technology was the very large scale integration (VLSI) where thousands of transistors could be integrated on a single chip. The main impact of VLSI was that, it was possible to produce a complete CPU or main memory or other similar devices on a single IC chip. This implied that mass production of CPU, memory etc. can be done at a very low cost. Let us take a brief review of the important breakthroughs of VLSI technologies.

Semiconductor Memories: Initially the IC technology was used for constructing processor, but soon it was realised that same technology can be used for construction of memory. The first memory chip was constructed in 1970 and could hold 256 bits. Although, the cost of this chip was high initially, gradually it is going down. The memory capacity per chip has increased as: 1k, 4k, 16k, 64k, 256k and 1M bytes or even more. Now, a standard PC sold in the market has a memory of 256MB or more. These memories can be random access memory (RAM), read only memory (ROM) or erasable, programmable read only memory (EPROM).

Microprocessors: Keeping pace with the electronics as more and more components were fabricated on a single chip, fewer chips were needed to construct a single processor. Intel in 1971 achieved the breakthrough of putting all the components on a single chip. The single chip processor is known as a microprocessor. The Intel 4004 was the first microprocessor. It was a primitive microprocessor designed for a specific application. Intel 8080, which came in 1974, was the first general-purpose microprocessor. It was an 8-bit microprocessor. Motorola is another manufacturer in this area. At present 32- and 64-bit general-purpose microprocessors are already in the market. For example Intel 486 is a 32-bit processor, similarly Motorola's 68000 is a 32 bit microprocessor. Pentium, which was developed by Intel in 1993, processes integers as long as 64 bits. The VLSI technology is still evolving more and more powerful microprocessors and more storage space now is being put in a single chip.

In Fig. 6, we depict the evolution of Intel microprocessor families, which started with 8-bit chip to develop to latest available 32/64-bit Pentium chips. The computers are

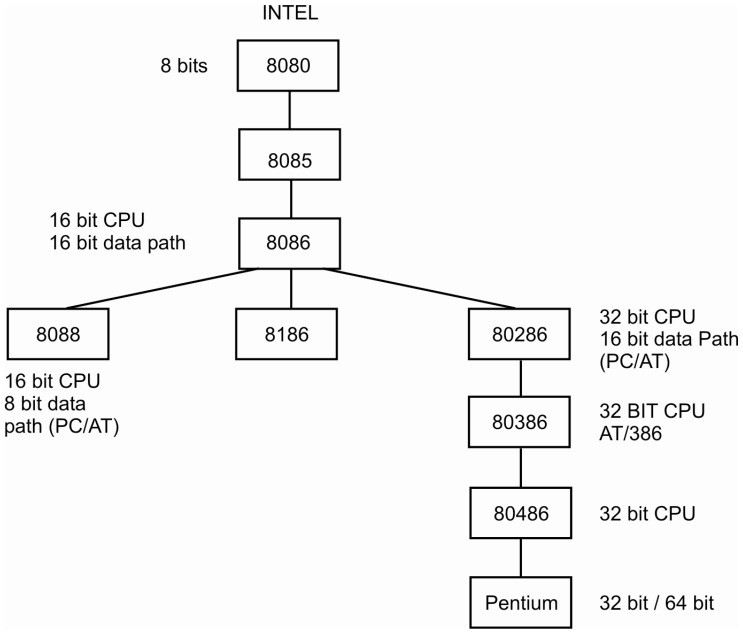


Fig. 6: Intel Microprocessor families.

broadly classified as micro-computers, mini-computers, mainframe computers and supercomputers. Although with development in technology the distinction between all these is becoming blurred, yet it is important to classify them as it is sometimes useful to differentiate the key elements and architecture among the classes.

a. Micro-computers

The CPU of a micro-computer is a microprocessor. The micro-computer originated in late 1970's. The first micro-computers were built around 8-bit microprocessor chips. What do we mean by an 8-bit chip? It means that the chip can retrieve instructions/ data from storage, manipulate, and process an 8-bit data at a time or we can say that the chip has a built-in 8-bit data transfer path. 8088 was a 8/16 bit chip i.e. an 8-bit path was used to move data between chip and primary storage (external path), but processing was done within the chip using a 16-bit path (internal path) at a time. 8086 was a 16/16 bit chip i.e. the internal and external paths both were 16 bit wide. Both these chips could support a primary storage capacity of up to 1 MB.

b. Mini-computer

The term mini-computer originated when it was realised that many computing tasks do not require an expensive contemporary mainframe computers but can be solved by a small, inexpensive computer. It is in the middle range of computing, between mainframes and micro computers. Earlier, mini-computers had distinct hardware and operating system features that separated them from the other two types. With advances in technology, the distinction between micro-computers and mini-computers is getting increasingly blurred.

With the advancement in technology the speed, memory size and other characteristics developed and the mini-computer was then used for various stand alone or dedicated applications. The mini-computer was then used as a multi-user system, which can be used by various users at the same time. This was actually the first conceptual realisation of computer networking, where the multiple users were connected using hardware. The concept started from the fact that the CPU performed the processing at much faster

rates than those required to entire new data and its display. The spare unused time of CPU could be used by other users. Gradually the architectural requirement of mini-computers grew and a 32-bit mini-computer, which was called super-mini, was introduced. The super-mini had more peripheral devices, larger memory and could support more users working simultaneously on the computer in comparison to previous mini-computers. This class of machines are now more commonly used as servers.

c. Mainframes

Mainframe computers are suited to big organisations, to manage high volume applications. They can run multiple operating systems at the same time. They are generally used in business applications. The term mainframe is being gradually replaced by the term **enterprise server**. Few of popular mainframe series are DEC, IBM, HP, ICL, MEDHA, Sperry, etc. Mainframes are also used as central host computers in distributed systems. Libraries of application programs developed for mainframe computers are much larger than those of the micro- or mini-computers because of their evolution over several decades as families of computing. All these factors and many more make the mainframe computers indispensable even with the popularity of micro-computers.

d. Supercomputers

The upper end of the state of the art mainframe machine is the supercomputer. Super computers are amongst the fastest machines in terms of processing speed and use multiprocessing techniques, where a number of processors are used to solve a problem. There are a number of manufacturers who dominate the market of supercomputers-CRAY (CRAY YMP, CRAY 2), ETA (CDC-ETA 10, ETA 20), IBM 3090 (with vector), NEC (NEC SX-3), Fujitsu (VP Series) and HITACHI (S Series) are same of them. Lately ranges of parallel computing products, which are multiprocessors sharing common buses, have been in use in combination with the mainframe supercomputers. The supercomputers are reaching up to speeds well over 25000 million arithmetic operations per second. India has also developed its indigenous supercomputer PARAM.

Supercomputers are mainly being used for number crunching problems such as weather forecasting, computational fluid dynamics, remote sensing, image processing, biomedical applications, etc.

E8) i) What is a general-purpose machine?

ii) What are the advantages of IC technology over discrete components?

The processor, memory and peripheral devices comprise the *hardware* part of the computer. However, all these parts function only because of the software incorporated in the computers. Let us now take brief account of computer software.

16.5 COMPUTER SOFTWARE

Computer software consists of sets of instructions that mould the raw arithmetic and logical capabilities of the hardware units to perform. Computer software can be broadly classified into two categories—**Systems Software** and **Application Software.**

Today, there are many languages available for developing program software. These languages are designed keeping in mind some specific areas of applications. Thus, some of the language may be good for writing system program/software while some other for application software.

i) **System Programming Languages:** System programs are designed to make the computer easier to use. An example of system software is an operating system, which consists of many other programs of controlling input/output devices, memory, processor, etc. To write an operating system, the programmer needs instruction to control the computer's circuitry (hardware part). For example, instructions that move data from one location of storage to a register of the processor. Today languages like C, C++ and C# languages are widely used to develop system software.

ii) **Application Programming Language:** Application programs are designed for specific computer applications, such as payroll processing, inventory control, word processing, etc. To write programs for pay-roll processing or other applications, the programmer does not need to control the basic circuitry of a computer. Instead the programmer needs instructions that make it easy to input data, produce output, do calculations and store and retrieve data. Programming languages that are suitable for such application programs support these instructions but not necessarily the types of instructions needed for development of system programs.

There are two main categories of application programs: **Business programs** and **scientific application programs**. Most programming languages are designed to be good for one category of application but not necessarily for the other, although there are some general-purpose languages that support both types. Business applications are characterised by processing of high volume data but call for simple calculations. Languages which are suitable for business program development must support high volume input, output and storage but need not support complex calculations. On the other hand, programming languages that are designed for writing scientific program contain very powerful instructions for calculations but rather poor instructions for input, output, etc. Amongst traditionally used programming languages, COBOL (Commercial Business Oriented Programming Language) is more suitable for business applications whereas FORTRAN (Formula Translation Language) and C-language are more suitable for scientific applications. Before we discuss more about language let us briefly look at the categories of software viz. system and application software.

16.5.1 System Software

Operating System

An operating system (OS) is the most important system software and is a must to operate a computer system. An operating system manages computer's resources very effectively, takes care of scheduling multiple jobs for execution and manages the flow of data and instructions between the input/output units and the main memory. Operating system became a part of computer software with the second-generation computers. Since then operating systems have undergone several revisions and modifications in order to achieve a better utilisation of computer resources. Advances in the field of computer hardware have also helped in the development of more efficient operating systems.

Language Translator

A language translator is a system software which translates a computer program written by a user into a machine understandable form.

Utilities

Utility programs are those which are very often requested by many application programs. A few examples are:

1) SORT/MERGE for sorting large volumes of data and merging them into a single sorted list.

2) Transfer program for transforming contents from one storage medium to another, e.g. disk to tape, tape to disk, etc.

Special Purpose Software

Special purpose program are those which extend the capability of operating systems to provide specialised services to application programs. A few examples are:

1) Spreadsheet software like Excel, Openoffice Calc, etc.
2) Data management software like Oracle, MySQL, SQLServer, etc.

These programs can also be used as stand alone application programs.

16.5.2 Application Software

Application software is written to enable the computer to solve a specific data processing task. There are two categories of application software:

- Software packages that are ready for use, and
- User application program.

A number of powerful application software packages, which do not require significant programming knowledge, have been developed. These are easy to learn and use as compared to the programming languages. Although these packages can perform many general and special functions, there are applications where these packages are not found adequate. In such cases, application program is written to meet the exact requirement. A user application program may be written using one of these packages or a programming language. Some important categories of software packages available are:

- Data Base Management Software
- Word Processing Desktop Publishing (DTP) and presentation Software
- Graphics Software
- Data Communication Software
- Statistical and Operational Research Software

Try the following exercises now.

E9) Explain the following terms in one or two sentences each:
(a) Operating Systems, and (b) Database Management Software

In the next section, we will discuss different categories of languages for writing software and their advantages and disadvantages.

16.6 CATEGORIES OF LANGUAGES

You can choose any language for writing a program according to the need. But a computer executes programs only after they are represented internally in binary form (sequences of 1s and 0s). Program written in any other languages must be translated to the binary representation of the instructions before they can be executed by the computer. Program written for a computer may be in one of the following categories of languages.

16.6.1 Machine Language

This is a sequence of instructions written in the form of binary numbers consisting of 1s, 0s to which the computer responds directly. You have used this language in Example 1. The machine language was initially referred to as code, although now the term code is used more broadly to refer to any program text.

As you have already learnt, an instruction prepared in any machine language will have at least two parts. The first part is the command or Operation, which tells the computer what function is to be performed. All computers have an operation code for each of its functions. The second part of the instruction is the operand or it tells the computer where to find or store the data that has to be manipulated.

Just as hardware is classified into generations based on technology, computer languages also have a generation classification based on the level of interaction with the machine. Machine language is considered to be the first generation language.

Advantage of Machine Language

It is faster in execution since the computer directly starts executing it.

Disadvantage of Machine Language

It is difficult to understand and develop a program using machine language. Anybody going through this program for checking will have a difficult task understanding what will be achieved when this program is executed. Nevertheless, the computer hardware recognises only this type of instruction code.

16.6.2 Assembly Language

When we employ symbols (letter, digit or special characters) for the operation part, the address part and other parts of the instruction code, this representation is called an assembly language program. This is considered to be the second generation language.

Machine and Assembly languages are referred to as low-level languages since the coding for a problem is at the individual instruction level.

Each machine has got its own assembly language, which is dependent upon the internal architecture of the processor.

An **Assembler** is a translator, which takes its input in the form of an assembly language program and produces machine language code as its output.

The following program is an example of an assembly language program for adding two numbers X and Y and storing the result in some memory location.

```
ld A, 7    ; Load register A with 7
ld B, 10   ; Load register B with 10
add A, B   ; A ← A + B
ld (100), A    ; Save the result in the location 100
halt    ; Halt process
```

From this program, it is clear that usage of mnemonics (in our example `ld, add, halt` are mnemonics) has improved the readability of our program significantly.

An assembly language program cannot be executed by a machine directly as it is not in a binary form. An assembler is needed in order to translate an assembly language program into the object code executable by the machine.

Advantage of Assembly Language

Writing a program in assembly language is more convenient than in machine language. Instead of binary sequence, as in machine language, it is written in the form of symbolic instructions. Therefore, it gives better readability than the machine language program.

Disadvantages of Assembly Language

Assembly language (program) is specific to particular machine architecture. Assembly languages are designed for specific make and model of a microprocessor. It means that assembly language programs written for one processor will not work on a different processor if it is architecturally different. That is why the assembly language program is *not portable*.

Assembly language program is not as fast as machine language since it has to be first translated into machine (binary) language code.

16.6.3 High-Level Languages

You must have already heard about the programming languages such as COBOL, FORTRAN, BASIC, PASCAL, JAVA, C, etc. They are called high-level programming languages. The time and cost of creating machine and assembly language was quite high. And this was the prime motivation for the development of high-level languages. The program shown below is written in BASIC to obtain the sum of two numbers.

```
10 LET X = 7
20 LET Y=10
30 LET SUM=X+Y
40 PRINT SUM
50 END
```

The high-level source program must be translated first into the form that machine can understand. This is done by a software called **compiler** which takes the source code as input and produces an output in the machine language code of the machine on which it is to be executed.

During the process of translation, the compiler reads the source program statements one by one and checks the syntax (grammatical) errors. If there is any error, the computer generates a print-out of the errors it has detected. This action is known as **diagnostics.**

There is another type of software, which also does the translation. This is called an **interpreter**. The compiler and interpreter have different approaches to translation. Table 5 lists the differences between a Compiler and an Interpreter.

Table 5: Differences between a Compiler and an Interpreter

	Compiler	Interpreter
1.	Scans the entire program first and then translates it into machine code.	Translates the program line by line.
2.	Converts the entire program to machine code; when all the syntax errors are removed execution takes place.	Each time the program is executed every line is checked for syntax error and then converted to equivalent machine code.
3.	Slow for debugging (removal of mistakes from a program).	Good for fast debugging.
4.	Execution time is less.	Execution time is more.

Advantage of High-level Programming Language: There are four main advantages of high-level programming languages. These are:

i) **Readability:** Programs written in these languages are more readable than assembly and machine language.

ii) **Portability**: Programs could be run on different machines with little or no change. We can, therefore, exchange software leading to creation of program libraries.

iii) **Easy debugging**: Errors could easily be removed (debugged).

iv) **Easy Software development**: Software could be easily developed. Commands of programming language are similar to natural languages (English).

16.6.4 Fourth Generation of Programming Languages

The fourth generation of programming languages is not as clearly defined, as are the other earlier generations. Most people feel that a fourth generation language, commonly referred to as 4GL is a high level language that requires significantly fewer instructions to accomplish task than a third generation language does. Thus, a programmer should be able to write a program faster in 4GL than in third generation language.

Most third generation languages are procedural languages. This means that the programmer must specify the steps, that is the procedure, the computer has to follow in a program. By contrast, most fourth generation languages are non-procedural languages. The programmer does not have to give the details of procedure in the program but instead, specifies what is wanted. For example, assume that a programmer needs to display some data on a screen, such as the address of a particular employee (SANTOSH) from the personnel file. In a procedural language, the programmer would have to write a series of instructions in the following steps:

Step 1 : Get a record from the personnel file.

Step 2 : If this is the record for SANTOSH, display the address.

Step 3 : If this is not the record for SANTOSH, go to Step 1.

In a non-procedural language (4GL), however, the programmer would write a single instruction that says:

Get the address of SANTOSH from personnel file.

Major fourth generation languages are used to get information from files and data base, as in the above example and to display or pint the information. Fourth generation languages are mostly machine independent. Usually they can be used on more than one type of computer. They are mostly used for office automation or business applications, but not for scientific program. Some fourth generation languages are designed to be easily learnt and used by end users.

E10) State whether the following statements are True or False.

a) Assembly language programs are machine independent.

b) Machine language programs are machine independent.

c) High level languages are machine independent.

d) All programs are machine independent.

e) 4th generation languages are dependent on databases they use.

We close the section here. In the next section, we will summarise our discussion in this Unit.

16.7 SUMMARY

In this Unit we have discussed

1. the von Neumann architecture of computers;
2. the binary, octal and hexadecimal systems;
3. how to convert numbers from one system to another system;
4. the evolution of computers;
5. the various types of softwares available; and
6. the various categories of computer languages, their strengths and weaknesses.

16.8 SOLUTIONS/ANSWERS

E1) a)

Digit	Weight	Face value	Product
1	1	1	1
0	2	1	0
1	4	0	4
1	8	1	8
1	16	1	16
1	32	1	32
			61

b)

Digit	Weight	Face value	Product
3	1	3	3
4	8	4	32
6	64	6	384
4	512	4	2048
3	4096	3	12288
3	32768	3	98304
			113059

c)

Digit	Weight	Face value	Product
C	1	12	12
B	16	11	176
3	256	3	768
2	4096	2	8192
1	65536	1	65536
A	1048576	10	10485760
			10560444

E2)

a)

2	61	$\rightarrow 1$
2	30	$\rightarrow 0$
2	15	$\rightarrow 1$
2	7	$\rightarrow 1$
2	3	$\rightarrow 1$
	1	

b)

8	113059	$\rightarrow 3$
8	14132	$\rightarrow 4$
8	1766	$\rightarrow 6$
8	220	$\rightarrow 4$
8	27	$\rightarrow 3$
	3	

c)

16	10560446	$\rightarrow 12(=\text{E})$
16	660027	$\rightarrow 11(=\text{B})$
16	41251	$\rightarrow 3$
16	2578	$\rightarrow 2$
16	161	$\rightarrow 1$
	$10(=\text{A})$	

E3) Grouping in bunches of 3 and finding the value of each bunch, we have $\underbrace{11}_{3}\ \underbrace{110}_{6}\ \underbrace{101}_{5}$. So, the given number is 365_8 in octal.

Grouping in bunches of 4 and finding the value of each bunch, we have $\underline{1111}_{15}\underline{0101}_{5}$. So, the number is $F5_{16}$ in hexadecimal.

E4) $-10 = 1111\ 0110$ in 2s complement notation. $-17 = 0001\ 0010$ in 2's complement notation.

E5) $-3 = 1111\ 1100$ in 1's complement and 1111 1101 in 2's complement. -5 is 1111 1011 in 2's complement. We have worked out the multiplication below. Note that the multiplication is similar to usual multiplication. The first digit in the multiplier 1111 1011 is 1. So, we just copy the multiplicand in the first row of the answer. The next digit in the multiplicand is 0. We do nothing, but when we multiply by the third digit which is 1, we move two digits to the left instead of 1 digit. When we copy the multiplier, we discard all the digits beyond the 8th digit. We then add each column to get the answer.

1	1	1	1	1	1	0	1
1	1	1	1	1	0	1	1
1	1	1	1	1	1	0	1
1	1	1	1	0	1		
1	1	1	0	1			
1	1	0	1				
1	0	1					
0	1						
1							
0	0	0	0	1	1	1	1

Since 0000 1111 is 15 in binary, the 2's complement arithmetic works for multiplication too!

E6) i) T ii) F iii) T iv) T v) T

E7) 1) Move data from memory location A to temporary register 1.
2) Move data from memory location B to AC.
3) Move data from AC to location A.
4) Move data from temporary register 1 to memory location B.

E8) i) A machine, which can be used for variety of applications and is not modelled only for specific applications. Von Neumann machines are general-purpose machines since they can be programmed for any general application, while a microprocessor based control systems are not general-purpose machines as they are specifically modelled as control systems.

ii) Low cost; increased operating speed; reduction in size of the computers; reduction in power and cooling requirements; and more reliable performance.

Photo Credits.
Eniac Computer from
`http://www.library.upenn.edu/exhibits/rbm/mauchly/jwm0-1.html`