CHAPTER 5 COMPUTER SOFTWARE

5.1 INTRODUCTION

A computer system consists of hardware and software. The computer hardware cannot perform any task on its own. It needs to be instructed about the tasks to be performed. Software is a set of programs that instructs the computer about the tasks to be performed. Software tells the computer how the tasks are to be performed; hardware carries out these tasks. Different sets of software can be loaded on the same hardware to perform different kinds of tasks

user can use the same computer hardware for writing a report or for running a payroll program. The components like monitor, keyboard, processor, and mouse, constitute the hardware (Figure 5.1). In this chapter, we will discuss the different categories of computer software.

Figure 5.1 Making diagrams using hardware and software

6.1 TYPES OF SOFTWARE

Software can be broadly classified in two categories:

1. System Software, and 2. Application Software.

System software provides the basic functions that are performed by the computer. It is necessary for the functioning of a computer. Application software is used by the users to perform specific tasks. The user may choose the appropriate application software, for performing a specific task, which provides the desired functionality. The system software interacts with hardware at one end and with application software at the other end. The application software

interacts with the system software and the users of the computer. Figure 5.2 shows the hierarchy of software, hardware and users.

Figure 5.2 Software hierarchy

5.2 SYSTEM SOFTWARE

System software provides basic functionality to the computer. System software is required for the working of computer itself. The user of computer does not need to be aware about the functioning of system software, while using the computer. For example, when you buy a computer, the system software would also include different device drivers. When you request for using any of the devices, the corresponding device driver software interacts with the hardware device to perform the specified request. If the appropriate device driver for any device, say a particular model of a printer, is installed on the computer, the user does not need to know about the device driver, while printing on this printer.

The *purposes of the system software* are:

- To provide basic functionality to computer,
- To control computer hardware, and
- To act as an interface between *user*, *application software* and *computer hardware*.

On the basis of their functionality, system software may be broadly divided into two categories
(Figure 5.3) as follows—

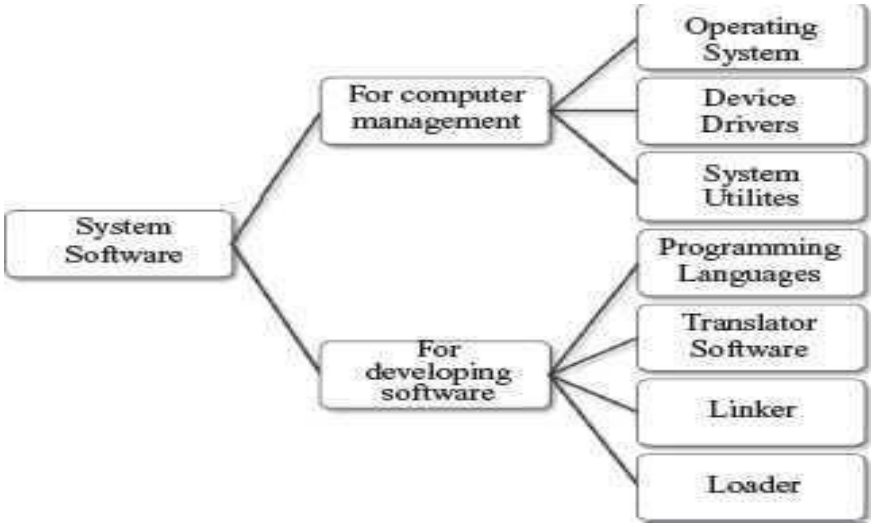

Figure 5.3 System software

- *System software for the management and functionality of computer relates* to the functioning of different components of the computer, like, processor, input and output devices etc. System software is required for managing the operations performed by the components of computer and the devices attached to the computer. It provides support for various services, as requested by the application software. Operating system, device drivers, and system utilities constitute the system software for management of computer and its resources.
- System software for the *development of application software* provides services required for the development and execution of application software. System software provides the software tools required for the development of application software. The programming language software, translator software, loader, and linker are also categorized as system software, and are required for the application software development.

5.3.1 Operating System

Operating System (OS) is an important part of a computer. OS intermediates between the user of a computer and the computer hardware. Different kinds of application software use specific hardware resources of a computer like CPU, I/O devices and memory, as needed by the application software. OS controls and coordinates the use of hardware among the different application software and the users. It provides an interface that is convenient for the user to use, and facilitates efficient operations of the computer system resources. The key functions of OS are—

- It provides an environment in which users and application software can do work.
- It manages different resources of the computer like the CPU time, memory space, file storage, I/O devices etc. During the use of computer by other

programs or users, operating system manages various resources and allocates them whenever required, efficiently.

- It controls the execution of different programs to prevent occurrence of error.
- It provides a convenient interface to the user in the form of commands and graphical
  interface, which facilitates the use of computer.

Some available operating systems are Microsoft Disk Operating System (MS-DOS), Windows 7, Windows XP, Linux, UNIX, and Mac OS X Snow Leopard.

5.3.2 Device Driver

A device driver acts as a translator between the hardware and the software that uses the devices. In other words, it intermediates between the device and the software, in order to use the device.

Some devices that are commonly connected to the computer are—keyboard, mouse, hard disk, printer, speakers, microphone, joystick, webcam, scanner, digital camera, and monitor. For proper working of a device, its corresponding device driver must be installed on the computer. For example, when we give a command to read data from the hard disk, the command is sent to the hard disk driver and is translated to a form that the hard disk can understand. The device driver software is typically supplied by the respective device manufacturers.

Programmers can write the higher-level application code independently of whatever specific hardware devices it will ultimately use, because code and device can interface in a standard way, regardless of the software superstructure, or of the underlying hardware. Each version of a device, such as a printer, requires its own hardware-specific specialized commands. In contrast, most applications instruct devices (such as a file to a printer) by means of high level generic commands for the device, such as PRINTLN (print a line). The device-driver accepts these generic high-level commands and breaks them into a series of low-level, device-specific commands, as required by the device being driven.

Nowadays, the operating system comes preloaded with some commonly used device drivers, like the device driver for mouse, webcam, and keyboard. The device drivers of these devices are pre- installed on the computer, such that the operating system can automatically detect the device when it is connected to the computer. Such devices are called *plug and play devices*. In case the computer does not find the device driver, it prompts the user to insert the media (like a CD which contains the corresponding device driver) provided along with the device. Most device manufacturers, host the device drivers for their devices on their companies' websites; users can download the relevant driver and install it on their computer.

Each device has its own device driver (Figure 5.4).
Whenever a new device is connected to a computer, its device driver has to be loaded in the computer·s memory, to enable use of the device. When you buy a new printer, you get the device driver CD with it. You must install the device driver on your computer, to use the new printer. Each printer comes with its own device driver. If you replace your old printer with a new model, you need to install the device driver for the new printer.
Device drivers can be *character* or *block device* drivers. Character device drivers are for character based devices like keyboard, which transfer data character by character. Block device driver are for devices that transfer data as a block, like in hard disk.

Figure 5.4 Device driver (i) CD of a printer (ii) Sony audio recorder

5.3.3 System Utilities

System utility software is required for the maintenance of computer. System utilities are used for supporting and enhancing the programs and the data in computer. Some system utilities may come embedded with OS and others may be added later on. Some examples of system utilities are:

- *Anti-virus* utility to scan computer for viruses (Figure 5.5). *Data Compression* utility to compress the files.

Figure 5.5 Antivirus software on a computer

- *Cryptographic* utility to encrypt and decrypt files.

- *Disk Compression* utility to compress contents of a disk for increasing the capacity of a disk.
- *Disk Partitioning* to divide a single drive into multiple logical drives. Each drive is then treated as an individual drive and has its own file system. Figure 5.5 shows a hard disk with three partitions.
- *Disk Cleaners* to find files that have not been used for a long time. It helps the user to decide what to delete when the hard disk is full.

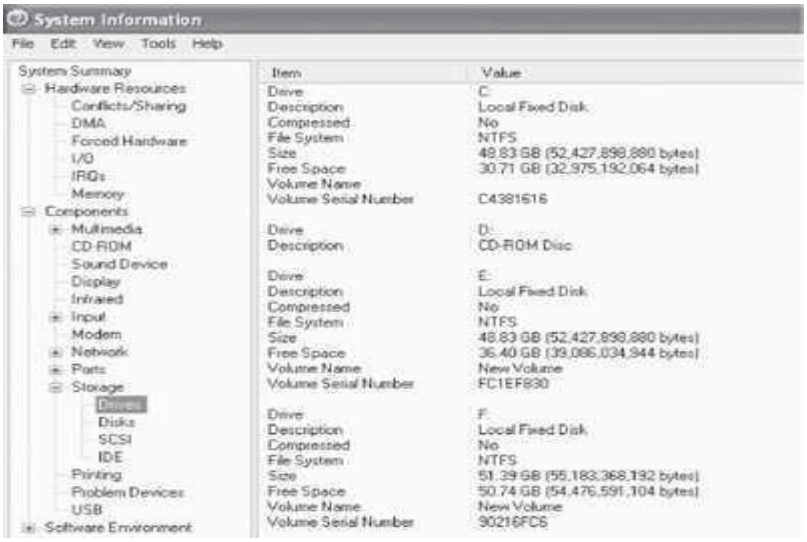

Figure 5.5 A hard disk having three partitions—C, E, and F

- *Backup Utility* to make a copy of all information stored on the disk. It also restores the backed up contents in case of disk failure.
- *System Profiling Utility* provides detailed information about the software installed on the computer and the hardware attached to it.
- *Network Managers* to check the computer network and to log events.

The system utilities on a computer working on Windows XP OS can be viewed by clicking <Start><All Programs><Accessories><System Tools>. Figure 5.7 shows system tools in Windows XP.

Figure 5.7 Some system tools in Windows XP

5.3.4 Programming Languages

A Programming Language consists of a set of vocabulary and grammatical rules, to express the computations and tasks that the computer has to perform. Programming languages are used to write a program, which controls the behavior of computer, codify the algorithms precisely, or enables the human-computer interface. Each language has a unique set of keywords (words that it understands) and a special syntax for organizing program instructions. The programming language should be understood, both by the programmer (who is writing the program) and the computer. A computer understands the language of 0·s and 1·s, while the programmer is more comfortable with English-like language. Programming Language usually refers to high-level languages like COBOL, BASIC, FORTRAN, C, C++, Java etc. Programming languages fall into three categories (Figure 5.8):

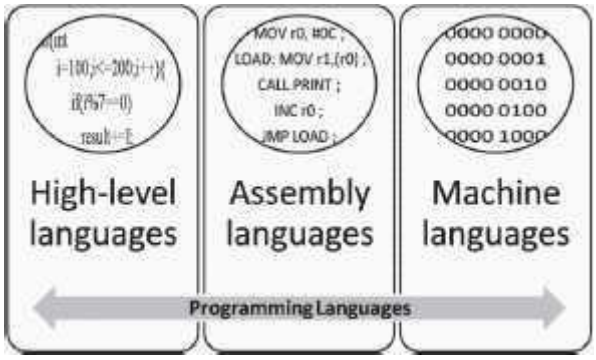

Figure 5.8 Programming languages

- *Machine Language* is what the computer can understand but it is difficult for the programmer to understand. Machine languages consist of numbers only. Each kind of CPU has its own unique machine language.
- *Assembly Language* falls in between machine language and high-level language. They are similar to machine language, but easier to program in, because they allow the programmer to substitute names for numbers.
- High-level Language is easier to understand and use for the programmer but difficult for the computer.

Regardless of the programming language used, the program needs to be converted into machine language so that the computer can understand it. In order to do this a program is either compiled or interpreted.

Figure 5.9 shows the hierarchy of programming languages. The choice of programming language for writing a program depends on the functionality required from the program and the kind of program to be written. Machine languages and assembly languages are also called *low-level languages*, and are generally used to write the system software. Application software is usually written in *high-level* languages. The program written in a programming language is also called the *source code*.

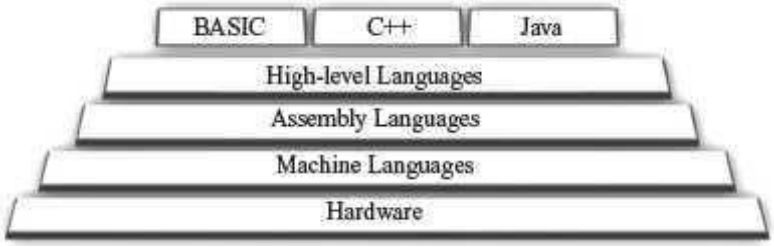

Figure 5.9 A program in machine language

5.4.3.1 Machine Language

A program written in machine language is a collection of binary digits or bits that the computer reads and interprets. It is a system of instructions and data executed directly by a computer·s CPU. It is also referred to as machine code or object code. It is written as strings of 0's and 1·s, as shown in Figure 5.10. Some of the features of a program written in machine language are as follows:

```
00000000101000010000000000011000
00000000100011100001100000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

Figure 5.10 Machine language code

- The computer can understand the programs written in machine language directly. No translation of the program is needed.
- Program written in machine language can be executed very fast (Since no translation is required).
- Machine language is defined by the hardware of a computer. It depends on the type of the processor or processor family that the computer uses, and is thus machine-dependent. A machine- level program written on one computer may not work on another computer with a different processor.
- Computers may also differ in other details, such as memory arrangement, operating systems, and peripheral devices; because a program normally relies on such factors, different computer may not run the same machine language program, even when the same type of processor is used.
- Most machine-level instructions have one or more opcode fields which specify the basic instruction type (such as arithmetic, logical, jump, etc), the actual operation (such as add or compare), and some other fields.
- It is difficult to write a program in machine language as it has to be written in binary code. For e.g., 00010001 11001001. Such programs are also difficult to modify.
- Since writing programs in machine language is very difficult, programs are hardly written in machine language.

5.3.4.2 Assembly Language

A program written in assembly language uses symbolic representation of machine codes needed to program a particular processor (CPU) or processor family. This representation is usually defined by the CPU manufacturer, and is based on abbreviations (called mnemonics) that help the programmer remember individual instructions, registers, etc. Small, English-like representation is used to write the program in assembly language, as shown in Figure 5.11. Some of the features of a program written in assembly language are as follows:

```
MOV   B, A
MVI   C, 06H
LXI   H, XX50H
ADD   M
JNC   NXTITM
INR   B
INX   H
DCR   C
JNZ   NXTBIT
```

Figure 5.11 Assembly language code

Assembly language programs are easier to write than the machine language programs, since assembly language programs use short, English-like representation of machine code. For e.g.:

ADD 2, 3

LOAD A

SUB A, B

The program written in assembly language is the source code, which has to be converted into machine code, also called object code, using translator software, namely, assembler. Each line of the assembly language program is converted into one or more lines of machine code. Hence assembly language programs are also machine-dependent.
Although assembly language programs use symbolic representation, they are still difficult to write. Assembly language programs are generally written where the efficiency and the speed of program are the critical issues, i.e. programs requiring high speed and efficiency.

5.3.4.3 High-level Language

A program in a high-level language is written in English-like language. Such languages hide the details of CPU operations and are easily portable across computers. A high-level language isolates the execution semantics of computer architecture from the specification of the program, making the process of developing a program simpler and more understandable with respect to assembly

and machine level languages. Some of the features of a program written in high-level language are as follows:

Programs are easier to write, read or understand in high-level languages than in machine language or assembly language. For example, a program written in C++ is easier to understand than a machine language program (Figure 5.12).

Programs written in high-level languages is the source code which is converted into the object code (machine code) using translator software like interpreter or compiler.

A line of code in high-level program may correspond to more than one line of machine code. Programs written in high-level languages are easily portable from one computer to another.

Different Generations of Programming Languages

In addition to the categorization of programming languages into machine language, assembly language, and high-level language, programming languages are also classified in terms of generations in which they have evolved. Table 5.1 shows the classification of programming languages based on generations.

Translator Software

Translator software is used to convert a program written in high-level language and assembly language to a form that the computer can understand. Translator software converts a program written in assembly language, and high-level language to a machine-level language program (Figure 5.13). The translated program is called the *object code*. There are three different kind of translator software:

First Generation	Machine language
Second Generation	Assembly language
Third Generation	C, COBOL, Fortran, Pascal, C++, Java, ActiveX (Microsoft) etc.
Fourth Generation	.NET (VB.NET, C#.NET etc.) Scripting language (Javascript, Microsoft Frontpage etc.)
Fifth Generation	LISP, Prolog

Table 5.1 Generations of programming languages

Figure 5.13 Translator software

- Assembler,
- Compiler, and Interpreter.

Assembler converts a program written in assembly language to machine language. Compiler and interpreter convert a program written in high-level language to machine language. Let's now discuss, briefly, the different kinds of translator software.

5.3.4.4 Assembler

Assembly language is also referred to as a symbolic representation of the machine code. Assembler is a software that converts a program written in assembly language into machine code (Figure 5.14). There is usually a one-to-one correspondence between simple assembly statements and machine language instructions. The machine language is dependent on the processor architecture, though computers are generally able to carry out the same functionality in different ways. Thus the corresponding assembly language programs also differ for different computer architectures.

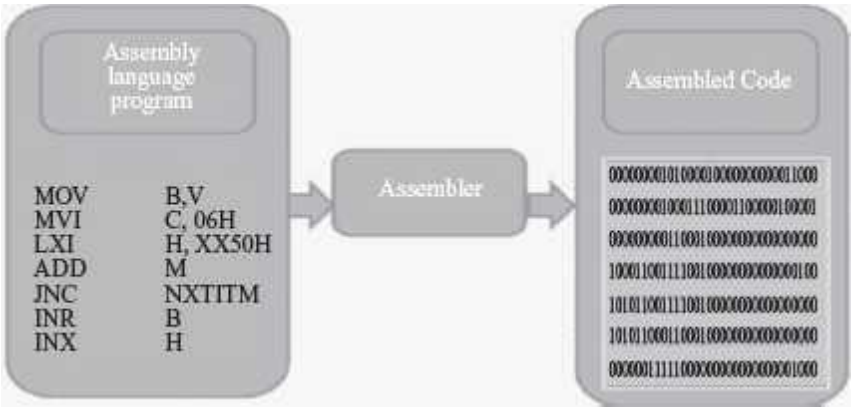

Figure 5.14 Assembler

5.3.4.5 Compiler

A program written in a high-level language has to be converted to a language that the computer can understand, i.e. *binary form*. Compiler is the software that translates the program written in a high-level language to machine language. The program written in high-level language is referred to as the *source code* and compiled program is referred as the *object code*. The object code is the *executable code*, which can run as a stand-alone code. It does not require the compiler to be

present during execution. Each programming language has its own compiler. Some languages
that use a compiler are C++, COBOL, Pascal, and FORTRAN. In some languages, compilation using the compiler and linking using the linker are required for creating the executable object code.

The compilation process generally involves two parts—breaking down the source code into small pieces and creating an intermediate representation, and, constructing the object code for the intermediate representation. The compiler also reports syntax errors, if any, in the source code.

5.3.4.6 Interpreter

The purpose of interpreter is similar to that of a compiler. The interpreter is used to convert the high-level language program into computer-understandable form. However, the interpreter functions in a different way than a compiler. Interpreter performs line-by-line execution of the source code during program execution. Interpreter reads the source code line-by-line, converts it into machine understandable form, executes the line, and then proceeds to the next line. Some languages that use an interpreter are BASIC and Python.

Difference Between a Compiler and An Interpreter: Compiler and Interpreter are used to convert a program written in high-level language to machine language; however, they work differently. The key differences between a compiler and an interpreter are as follows:

- Interpreter looks at a source code line-by-line. Compiler looks at the entire source code.
- Interpreter converts a line into machine executable form, executes the line, and proceeds with the next line. Compiler converts the entire source code into object-code and creates the object code. The object code is then executed by the user.
- For a given source code, once it is compiled, the object code is created. This object code can be executed multiple number of times by the user. However, interpreter executes line-by-line, so executing the program using an interpreter means that during each execution, the source code is first interpreted and then executed.
- During execution of an object code, the compiler is not required. However, for interpretation, both interpreter and the source code is required during execution (because source code is interpreted during execution).
- Since interpreter interprets line-by-line, the interpreted code runs slower than the compiled code.

5.4 APPLICATION SOFTWARE

The software that a user uses for accomplishing a specific task is the *application software*. Application software may be a single program or a set of programs. A

set of programs that are written for a specific purpose and provide the required functionality is called software package. Application software is written for different kinds of applications—graphics, word processors, media players, database applications, telecommunication, accounting purposes etc.

Some examples of application software packages (Figure 5.17) are as follows:

- *Word Processing Software:* For writing letter, reports, documents etc. (e.g. MS-WORD). *Image Processing Software:* For assisting in drawing and manipulating graphics (e.g. Adobe Photoshop).
- *Accounting Software:* For assisting in accounting information, salary, tax returns (Tally software).

Figure 5.17 Some application software

- *Spreadsheet Software:* Used for creating budget, tables etc. (e.g. MS-Excel).
- *Presentation Software:* To make presentations, slide shows (e.g. MS-PowerPoint)
- *Suite of Software having Word Processor, Spreadsheet and Presentation Software:* Some examples are MS-Office, Google Docs, Sun Openoffice, Apple iWork.
- CAD/CAM *Software:* To assist in architectural design. (e.g. AutoCAD, Autodesk)
- *Geographic Information Systems:* It captures, stores, analyzes, manages, and presents data, images and maps that are linked to different locations. (e.g. ArcGIS)
- *Web Browser Software:* To access the World Wide Web to search documents, sounds, images etc. (e.g. Internet Explorer, Netscape Communicator, Chrome).

5.5 SOFTWARE ACQUISITION

Different kinds of software are made available for use to users in different ways. The user may have to purchase the software, can download for free from the Internet, or can get it bundled along with the hardware. Nowadays with the advent of Cloud computing, many application software are also available on the cloud for use through the Internet, e.g. Google Docs. The different ways in which the software are made available to users are:

- **Retail Software** is off-the-shelf software sold in retail stores. It comes with printed manuals and installation instructions. For example, Microsoft Windows operating system. **OEM Software stands** for "Original Equipment Manufacturer" software. It refers to software which is sold, and bundled with hardware. Microsoft sells its operating system as OEM software to hardware dealers. OEM software is sold at reduced price, without the manuals, packaging and installation instructions. For example, Dell computers are sold with the "Windows 7" OS pre-loaded on them.
- **Demo Software** is designed to demonstrate what a purchased version of the software is capable of doing and provides a restricted set of features. To use the software, the user must buy a fully- functional version.
- **Shareware** is a program that the user is allowed to try for free, for a specified period of time, as defined in the license. It is downloadable from the Internet. When the trial period ends, the software must be purchased or uninstalled.
- **Freeware** is software that is free for personal use. It is downloadable from the Internet. The commercial use of this software may require a paid license. The author of the freeware software is the owner of the software, though others may use it for free. The users abide by the license terms, where the user cannot make changes to it, or sell it to someone else.
- **Public Domain Software** is free software. Unlike freeware, public domain software does not have a copyright owner or license restrictions. The source code is publicly available for anyone to use. Public domain software can be modified by the user.

Open-Source Software is software whose source code is available and can be customized and altered within the specified guidelines laid down by the creator. Unlike public domain software, open-source software has restrictions on their use and modification, redistribution limitations, and copyrights. Linux, Apache, Firefox, OpenOffice are some examples of open-source software.