3. Systems Software

Introduction

System software refers to the files and programs that makes up the computer's operating system. System files consist of libraries of functions, system services, drivers for printers, other hardware, system preferences and various configuration files.

System Software consist of the following programs:

- Assemblers
- Compilers
- File Management Tools
- System Utilities
- Debuggers

The system software is installed on the computer when the operating system is installed. The software can be updated by running programs such as "Windows Update" for Windows or "Software Update" for Mac OS X. Application programs are used by end user, whereas system software is not meant to be run by the end user. For example, while Web browser like Internet Explorer are used by users every day, they don't have to use an assembler program (unless the user is a computer programmer).

System software runs at the most basic level of the computer and hence it is called "low-level" software. User interface is generated by the system software and allows the operating system to interact with the hardware.

System software is the interface between the computer hardware and the application software.

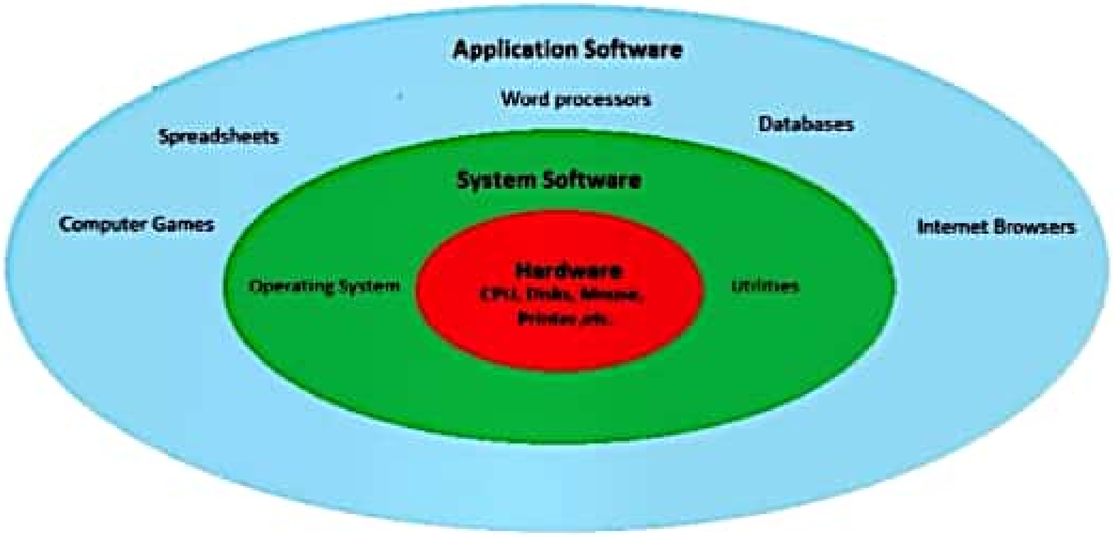

Figure 3.1 : System Software

3.1 Assembler

An assembler is a program that converts basic computer instructions into a pattern of bits. These bits are used by computer's processor to perform its basic operations. Some people call these instructions as assembler language and others use the term assembly language. This is how it works:

Most computers come with a specified set of very basic instructions that corresponds to the basic machine operations that the computer can perform. For example, a "Load" instruction moves a string of bits from a location in the processor's memory to a special holding place called a register. In computer architecture, a processor register is a small amount of storage available on the CPU and the contents of register can be accessed more quickly. Most of the modern computer architectures operate on the principle of shifting data from main memory into registers, operate on the data and then move the result back into main memory. Assuming the processor has at least eight numbered registers, the following instruction would move the value (string of bits of a certain length) at memory location 2000 into the holding place called register 8:

```
L 8,2000
```

- The programmer can write a program using a sequence of assembler instructions.
- These assembler instructions, which is also known as the source code or source program, is then specified to the assembler program when it is started.
- Each program statement in the source program is taken by the assembler program and a corresponding bit stream or pattern (a series of 0's and 1's of a given length) is generated.
- The output of the assembler program is called the object code or object program relative to the input source program. The object program consisting of a sequence of 0's and 1's is also known as machine code.
- This object program can be run (or executed) whenever required.

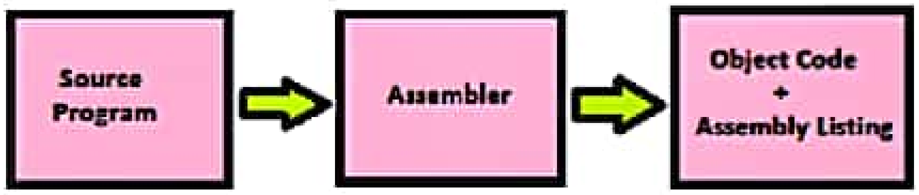

Figure 3.2 : Assembler

3.2 Assembly Language

The native language of the computer is assembly language. The processor of the computer understands machine code (consisting of ones and zeroes). In order to write the machine code program, it has to be first written in assembly language and then an assembler is used to convert the assembly language program to machine code.

Assembly language program consists of mnemonic codes which are easy to remember as they are similar to words in the English language. It consists of mnemonic codes for each of the different machine code instructions that the machine understands. A mnemonic is an abbreviation of the actual instruction. It is a programming code that is easy to remember because the codes resemble the original words, for example, ADD for addition and SUB for subtraction. Examples of assembly language program:

```
MOV EAX,2 ; set eax register to 2(eax = 2)

SHL EAX,4 : shift left value in the register

MOV ECX,16 ; set ecx register to 16

SUB EAX,EBX ; substracts ecx from eax
```

An assembler converts this set of instructions into a series of ones and zeros, also known as a executable program, that the machine can understand.

Advantages of Assembly language

- Assembly language can be optimized extremely well as it is extremely low level language. Therefore assembly language is used in application which require utmost performance.
- Assembly language can be used for communication with the machine at the hardware level and hence it is often used for writing device drivers.
- Another advantage of assembly language is the size of the resulting programs. Since conversion from a higher level by a compiler is not required, the resulting programs can be exceedingly small.

3.2.1 Assembly Language Programming

The assembler is a program which converts the assembly language source program into a format that can be run on the processor. The machine code instruction containing binary or hex value is replaced by a mnemonic.

Example

Binary	Hex	Mnemonic	Description
1001111	4F	CLRA	Clears the A accumulator
110110	36	PSHA	Saves A accumulator on stack
1001101	4D	TSTA	Tests A accumulator for 0

Figure 3.3 : Assembly Language Programming

- Mnemonics are easy to understand than hex or binary values.
- It is less likely to make an error.
- Mnemonics are easy to remember.

Assembly language statements are written one on each line. A machine code program consists of a sequence of assembly language statements in which each statement contains a mnemonic. A line of an assembly language program can contain the following four fields:

i. Label

ii. Opcode

iii. Operand

iv. Comments

The label field is optional. A label is an identifier or a text. Labels are used in programs to reduce reliance upon programmers remembering where data or code is located. A label is used to refer to the following:

- Memory location
- A data value
- The address of a program or a sub-routine or a portion of code.

The maximum length of a label differs for different type of assemblers. Some assemblers accept up to 32 characters long, others only four characters. When a label is declared, it is suffixed by a colon, and begins with a valid character (A..Z). Consider the following example.

LOOP: LDAA #24H

Here, the label LOOP is equal to the address of the instruction LDAA #24H. The label can be used as a reference in a program, as shown below

JMP LOOP

When the above instruction is executed, the processor will execute the instruction associated with the label LOOP, i.e.LDAA #24H. When a label is referenced later in a program, it is referenced without the colon suffix.

An advantage of using labels is that inserting or re-arranging code statements do not require re-working actual machine instructions. It only requires a simple re-assembly. In hand-coding, changes can take hours to perform.

The opcode field consists of a mnemonic. Opcode is the operation code, i.e., a machine code instruction. Opcode may also have additional information in the form of operands Operands are separated from the opcode by using a space.

Operands consists of additional information or data that the opcode requires. Operands are used to specify

- Constants or labels
- Immediate data
- Data present in another accumulator or register
- An address

Examples of operands

LDAA 0100H ; two byte operand

LDAA LOOP ; label operand

LDAA #1 ; immediate operand(i.e. constant value as operand)

The comment field is optional, and is used by the programmer to explain how the coded program works. The comments are prefixed by a semi-colon. Comments are ignored by the assembler when the instructions are generated from the source file.

3.3 Loaders and Linkers

In a computer operating system , a loader is a component that locates a program (which can be an application or a part of the operating system itself) in an offline storage (for eg. hard disk), loads it into main storage (also called random access memory), and gives that program control of the computer (i.e. allows it to execute its instructions).

A program that is loaded may itself contain components that are not initially loaded into main storage, but are loaded if and when their logic is required. In a multitasking operating system, a program that is known as a dispatcher juggles the computer processor's time among different tasks and calls the loader when a program associated with a task is not already in the main storage. (Program here, means a binary file that is the result of a programming language compilation or linkage editing or some other program preparation process.)

A linker, also known as link editor is a computer program that takes one or more object files generated by a compiler and combines them into a single executable program.

Computer programs comprise several parts or modules, but all these parts/modules need not be contained within a single object file. In such case they refer to each other by means of symbols. An object file can have three kinds of symbols:

- Defined symbols, which is called by other modules
- Undefined symbols, which call the other modules where the symbols are defined.
- Local symbols which are used internally within the object file to facilitate relocation.

For most compilers, each object file is the result of compiling one input source code file. If a program comprises multiple object files, then the linker combines these files into a unified executable program by resolving the symbols as it goes along.

Linkers can take objects from a collection called a runtime library. A runtime library is a collection of objects files which will contain machine code for any external function used by the program file used by the linker. This machine code is copied by the linker into the final executable output. Some linkers do not include the entire library in the output, they only include the symbols that are referenced from other object files or libraries. The libraries exist for diverse purposes. The system libraries are usually linked in by default.

The linker also arranges the objects in a program's address space. This involves relocating code that assumes a specific base address to another base address. Since a compiler seldom knows where an object will reside, it assumes a fixed base location (for example, zero). The relocation of machine code may involve re-targeting of loads, stores and absolute jumps.

The executable output by the linker may need another relocation pass when it is finally loaded into memory (just before the execution). This relocation pass is usually omitted on hardware offering virtual memory in which every program is put into its own address space, and so there is no conflict even if all programs load at the same base address. This relocation pass may also be omitted if the executable is a position independent executable.

3.4 Compilers

A compiler is a special program that processes statements written in a particular programming language and turns them into machine language or code that a computer's processor uses. A programmer writes language statements in a language such as Pascal or C one line at a time using an editor. The file created contains the source statements or source code. The programmer then runs the appropriate compiler for the language, specifying the name of the file that contains the source statements.

When executing (running), the compiler first parses (or analyses) the language statements syntactically one after the other and then builds the output code in one or more successive stages or "passes" and makes sure that statements that refer to other statements are referred in the final code correctly. The output of the compilation is called object code or sometimes an object module. The object code is machine code that the processor can execute, one instruction at a time.

The Basic Structure of a Compiler

In the five stages of a compiler, the high level language is translated to a low level language which is generally closer to that of the target computer. Each stage of the compiler fulfils a single task and has one or more classic techniques for implementation. The following are the five stages of a compiler:

- Lexical Analyser: Analyses the Source Code, removes "white space" and comments, formats it for easy access by creating tokens, then tags language elements with type information and begins to fill the information in the SYMBOL TABLE. The Symbol Table is a data structure that contains information about symbols and groups of symbols in the program being translated.

- Syntactic Analyser: Analyses the tokenized code for structure, groups symbols into syntactic groups, tags groups with type information.

- Semantic Analyser: Analyses the Parsed Code for meaning, fills in assumed or any missing information and tags the groups with the meaning.

- Code Generator: Linearizes the qualified Code and produces the object code.

- Optimizer: Checks the object code to determine whether there are more efficient means of execution.

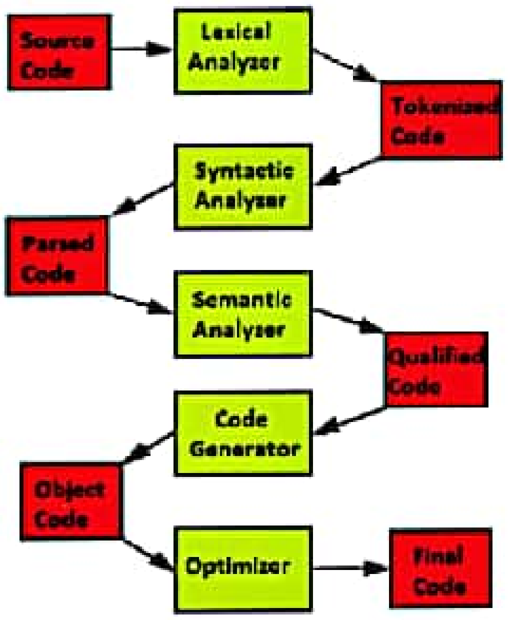

Figure 3.5 : Stages of a compiler

3.5 Interpreters

Interpreter is a program that executes instructions written in a high-level language. There are two methods to run programs written in a high-level language. The most common method is to compile the program and the other method is to pass the program through an interpreter.

An interpreter translates high-level instructions into an intermediate form and then executes it. In contrast to the above method, a compiler translates high-level instructions directly into machine language. The compiled programs generally run faster than interpreted programs. One of the advantage of using an interpreter is that it does not need to go through the compilation stage during which machine instructions are generated. This process of compilation can be time-consuming if the program is very long. The interpreter, on the other hand, immediately execute high-level programs. Hence interpreters are sometimes used during the development of a program in which the programmer wants to add small sections at a time and test them quickly. In addition to this, interpreters are often used in education because they allow students to program interactively.

Advantages of using interpreter

- Execution is done in a single stage. Compilation stage is not required.
- Alteration of code is possible during runtime.

- Really useful for debugging the codes.
- Helps in interactive code development.

Interpreter vs Compiler

- The primary difference between a compiler and interpreter is the way in which a program is executed. The compiler converts the source code to machine code and then saves it as an object code before creating an executable file for the same. The compiled program is executed directly using the machine code or the object code. But an interpreter does not convert the source code to an object code before execution.
- An interpreter executes the source code line by line and conversion to native code is performed line by line while execution is going on (at runtime). Hence the run time required for an interpreted program will be high compared to a compiled program.
- Even though the run time required for interpreted program is more, the execution using an interpreter has its own advantages. For example interpreted programs can modify themselves at runtime by adding or changing functions. Compiled program has to be recompiled fully even for the small modifications done in the program. But in the case of an interpreter there is no such problem (only the modified section needs to be recompiled). (Refer Fig 3.6)

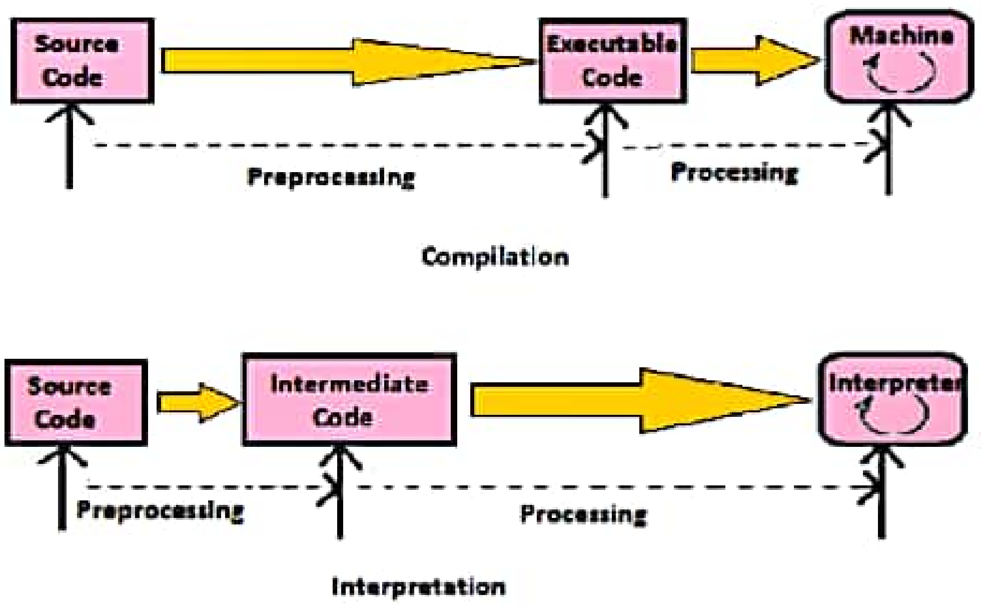

Figure 3.6 : Compiler vs Interpreter

Glossary

- A driver is software that helps the computer to communicate with hardware or devices. Without drivers, the hardware connected to the computer, for example, a webcam, will not work properly.
- An assembler is a computer program which translates from assembly language to an object file or machine language format.
- A compiler is a computer program that converts source code written in a particular programming language into another computer language.
- A debugger is a computer program that is used to test and debug other programs.
- Application software, also known as an application or an app, is computer software that helps the user to perform specific tasks. Examples of application software include enterprise software, office suites, graphics software and media players.