UMARU MUSA YAR'ADUA UNIVERSITY, KATSINA.
DEPARTMENT OF COMPUTER SCIENCE
FACULTY OF NATURAL AND APPLIED SCIENCES

CSC 3313

ADVANCED SYSTEM SOFTWARE

LECTURE NOTE

2024

INTRODUCTION

The term computer has been borrowed from compute that means to calculate. Whereas initially computers were used to perform arithmetic calculations at fast speed, now they are used in nearly every field. You can use computers for Banking Application, Word Processing, Desktop Publishing, Weather Forecasting, Control of Machine and Robots in Factory, Scientific Research, etc. In brief, a computer may be defined as a device that receives some kind of data, analyze it, and then applies a predefined set of instruction to it to produce some kind of output. The set of instructions given to the computer to perform various operations is called as the computer program. The process of converting the input data into the required output form with the help of the computer program is called as data processing. The computers are also referred to as data processors.

The Hardware:

- The hardware is the machinery itself. It is made up of the physical parts or devices of the computer system like the electronic Integrated Circuits (ICs), magnetic storage media and other mechanical devices like input devices, output devices etc. All these various hardware are linked together to form an effective functional unit. The various types of hardware used in the computers, has evolved from vacuum tubes of the first generation to Ultra Large Scale Integrated Circuits of the present generation.
- **The Software:**

  The computer hardware itself is not capable of doing anything on its own. It has to be given explicit instructions to perform the specific task. The computer program is the one which controls the processing activities of the computer. The computer thus functions according to the instructions written in the program. Software mainly consists of these computer programs, procedures and other documentation used in the operation of a computer system. Software is a collection of programs which utilize and enhance the capability of the hardware.

There are two broad categories of software:

- **System Software**
- **Application Software.**
  - **System Software** is a set of programs that manage the resources of a computer system. System Software is a collection of system programs that perform a variety of functions. Such as
    - File Editing
    - Resource Accounting
    - I/O Management,
    - Storage
    - Memory Management
    - Access management.

System Software can be broadly classified into three types as:

1. System Control Programs
2. System Support programs
3. System development Programs.

1. System Control Software: This refers to programs that coordinates and controls the computer programs, manage the storage & processing resources of the computer & perform other management & monitoring functions. The most important of these programs is the operating system. Other examples are database management systems (DBMS) & communication monitors.
2. System Support Software: Provide routine service functions to the other computer programs & computer users. Support programs do not make a direct contribution to performing the primary function of a computer system but rather serve to assist in the operation of the system. They perform tasks such as formatting disks, retrieving lost or damaged files, backing up important files, locating free space on a disk, libraries, performance monitors & job accounting etc. One of the most popular types of system support software is;

 Utility programs: Are programs that perform routine, repetitive tasks. The utility programs make it easier to use the computer.

1. System Development Software: This is the software packages and programs that assists programmers and system analysts in the design and development of application programs or information systems. E.g. web development, application generators, and language translators such as BASIC interpreter

Application Software:

It performs specific tasks for the computer user. Application software is a program which program written for, or, by, a user to perform a particular job. Languages already available for microcomputers include Clout, Q & A and Savvy ret rival (for use with Lotus 1-2-3).The use of natural language touches on expert systems, computerized collections of the knowledge of many human experts in a given field, and artificial intelligence, independently smart computer systems. General-purpose application software such as electronic spreadsheet has a wide variety of applications. Specific – purpose application s/w such as payroll & sales analysis is used for the application for which it is designed Application programmer writes these programs. Application programmer writes these programs.

How does the user interact with computer?

Generally computer users interact with application software. Application and system software act as interface between users & computer hardware. If an application & system software become more capable, people find computer easier to use.

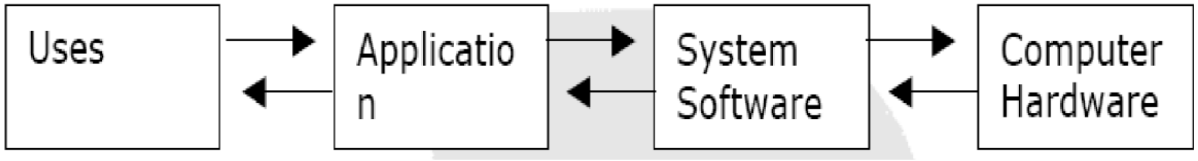


Figure 1: users interact with application software

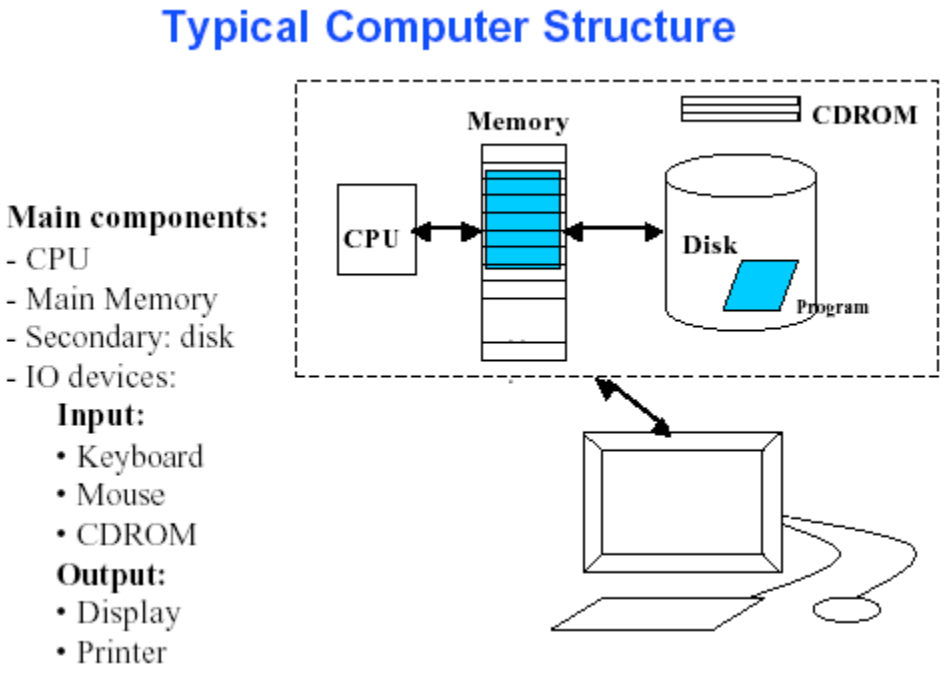


Figure 2: Computer structure

PROGRAM

A program is a series of logically and sequentially arranged instructions for solving problems. Programs are written using any of the programming languages. While a programmer is a trained specialist on computer programming languages and logic who can use any of the programming languages to direct computer on how to solve a given problem.

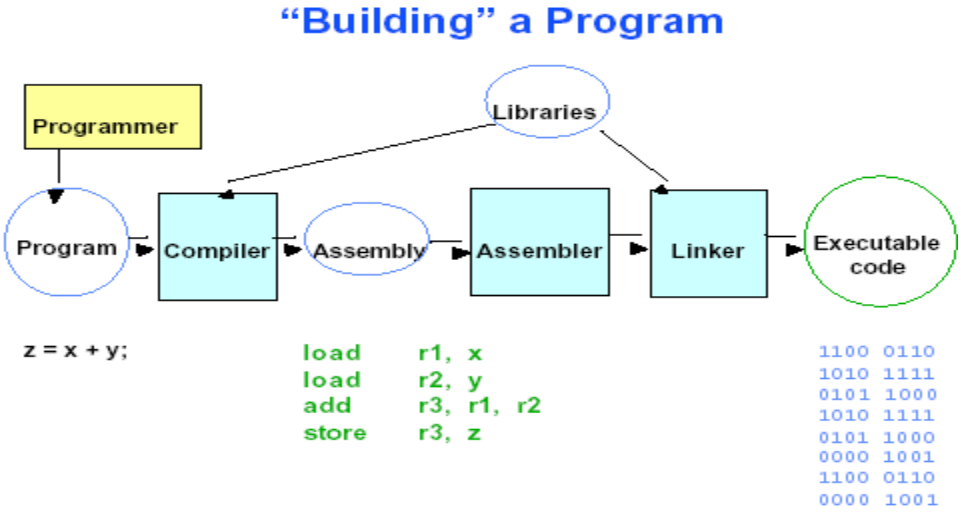


Figure 3: Building a program

Running the Program

- *Loader* puts the program into the computer memory
- Running the program is done by an Operating System command
- Input are read from the I/O devices and from Memory
- Output is written to I/O devices and to Memory

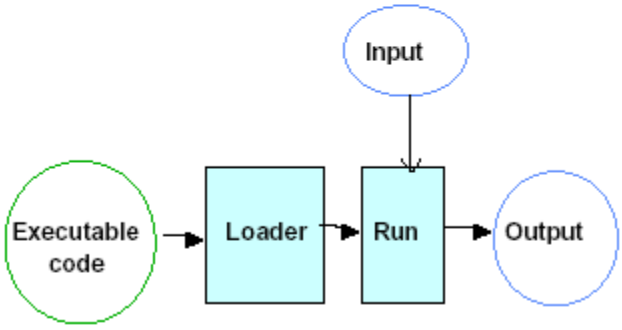


Figure 4: Running the program

Execution of a Program

- CPU executes the Instructions
- CPU reads Input Data from memory or I/O
- CPU stores the Output Data to memory or to I/O

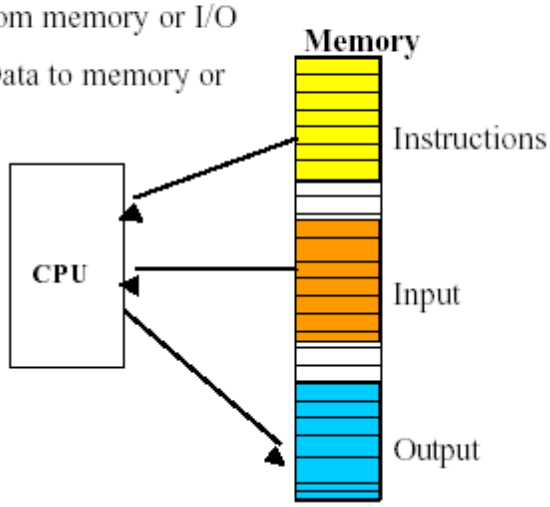


Figure 5: Execution of a program

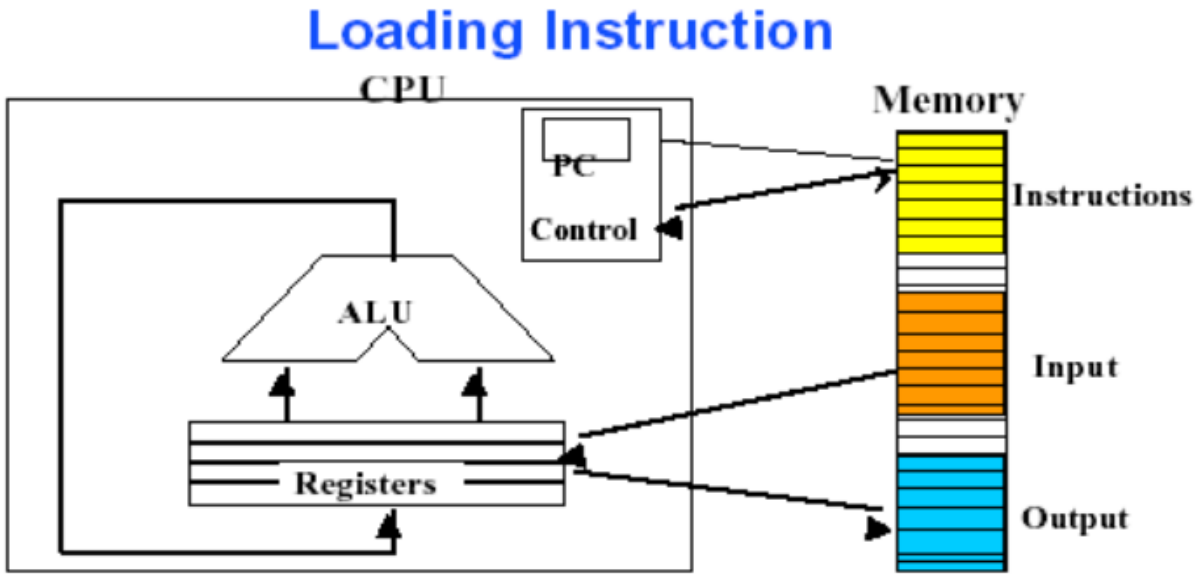


Figure 6: Loading Instruction

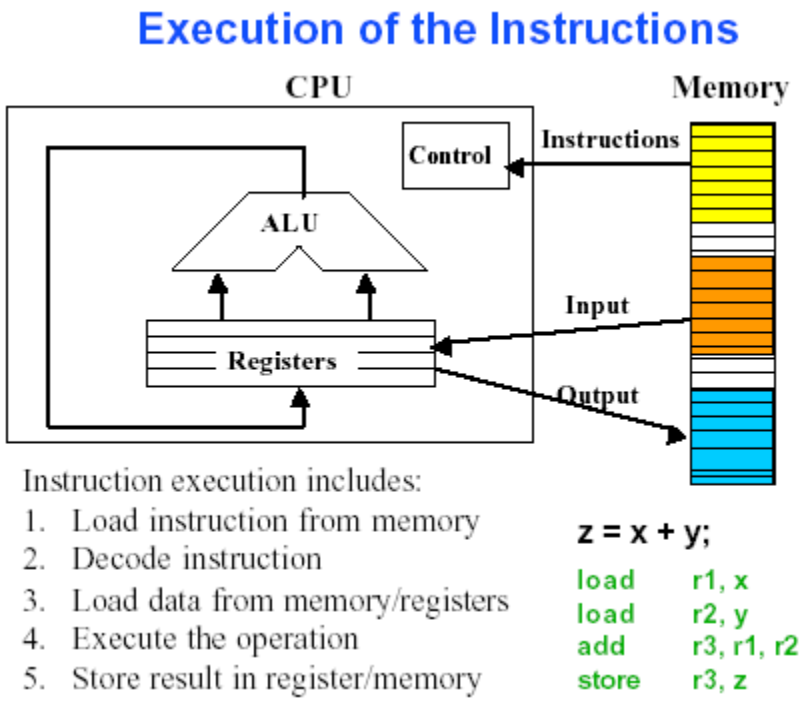


Figure 7: Execution of the Instruction

System Development Software:

System Development Software assists a programmer in developing & using an application program. E.g. Language Translators, Linkage Editors, and Application generators

Language Translators:

A language translator is a computer program that converts a program written in a procedural language (PL) such as BASIC into machine language that can be directly executed by the computer. Computers can execute only machine language programs. Programs written in any other language must be translated into a machine language load module, which is suitable for loading directly into primary storage. Subroutine or subprograms, which are stored on the system residence device to perform a specific standard function. E.g. if a program required the calculation of a square root, Programmer would not write a special program. He would simply call a square root, subroutine to be used in the program.

Translators for a low-level programming language were assemblers

Language processors: Language Processing activities arise due to the differences between the manner in which a software designer describes the ideas concerning the behavior of a software and the manner in which these ideas are implemented in a computer system. The interpreter is a language translator. This lead to many similarities between Translators and interpreters. From a practical viewpoint many differences also exist between them. The absence of a target program implies the absence of an output interface in the interpreter. Thus the language processing activities of an interpreter cannot be separated from its program execution activities. Hence we say that an interpreter 'executes' a program written in a procedural language (PL).

Problem Oriented and Procedure Oriented Languages:

The three consequences of the semantic gap mentioned at the start of this section are in fact the consequences of a specification gap. Software systems are poor in quality and require large amounts of time and effort to develop due to difficulties in bridging the specification gap. A classical solution is to develop a PL such that the PL domain is very close or identical to the application domain. Such PLs can only be used for specific applications; hence they are called problem-oriented languages. They have large execution gaps, however this is acceptable because the gap is bridged by the translator or interpreter and does not concern the software designer.

A procedure-oriented language provides general purpose facilities required in most application domains. Such a language is independent of specific application domains.

The fundamental language processing activities can be divided into those that bridge the specification gap and those that bridge the execution gap. We name these activities as:

1. Program generation activities and
2. Program execution activities.

1. A program generation activity aims at automatic generation of a program. The source languages specification language of an application domain and the target language is typically a procedure oriented PL.

2. A Program execution activity organizes the execution of a program written in a PL on computer system. Its source language could be a procedure-oriented language or a problem oriented language. **Program Generation**

The program generator is a software system which accepts the specification of a program to be generated, and generates a program in the target PL. In effect, the program generator introduces a new domain between the application and PL domains we call this the ***program generator domain***. The specification gap is now the gap between the application domain and the program generator domain. This gap is smaller than the gap between *the* application domain and the target PL domain. Reduction in the specification gap increases the reliability of the generated program. Since the generator domain is close to the application domain, it is easy for the designer or programmer to write the specification of the program to be generated. The harder task of bridging the gap to the PL domain is performed by the generator. This arrangement also reduces the testing effort. Proving the correctness of the pro-gram generator amounts to proving the correctness of the transformation. This would be performed while implementing the generator. To test an application generated by using the generator, it is necessary to only verify the correctness of the specification input to the program generator. This is a much simpler task than verifying correctness of the generated program. This task can be further simplified by providing a good diagnostic (i.e. error indication) capability in the program generator, which would detect inconsistencies in the specification.

It is more economical to develop a program generator than to develop a problem-oriented language. This is because a problem oriented language suffers a very large execution gap between the PL domain and the execution domain whereas the program generator has a smaller semantic gap to the target PL domain, which is the domain of a standard procedure oriented language. The execution gap between the target PL domain and the execution domain is bridged by the compiler or interpreter for the PL.

Program Execution: Two popular models for program execution are translation and interpretation.

Program translation: The program translation model bridges the execution gap by translating a program written in a PL, called the source program (SP), into an equivalent program in the machine or assembly language of the computer system, called the object program or target program (TP).

Characteristics of the program translation model are:

A program must be translated before it can be executed.

• The translated program may be saved in a file. The saved program may be executed repeatedly.

• A program must be retranslated following modifications.

Program interpretation

The interpreter reads the source program and stores it in its memory. During interpretation it takes a source statement, determines its meaning and performs actions which implement it. This includes computational and input-output actions. The CPU uses a program counter (PC) to note the address

of the next instruction to be executed. This instruction is subjected to the instruction execution cycle consisting of the following steps:

1. Fetch the instruction: Fetch the statement

2. Decode the instruction to determine the operation to be performed, and also its operands. Analyze the statement and determine its meaning, viz. the computation to be performed and its operands.

3. Execute the instruction.

At the end of the cycle, the instruction address in PC is updated and the cycle is repeated for the next instruction. Program interpretation can proceed in an analogous manner. Thus, the PC can indicate which statement of the source program is to be interpreted next. This statement would be subjected to the interpretation cycle, which could consist of the following steps:

1. Execute the meaning of the statement from this analogy, we can identify the following characteristics of interpretation: The source program is retained in the source form itself, i.e. no target program form exists, A statement is analyzed during its interpretation.

2. Comparison: A fixed cost (the translation overhead) is incurred in the use of the program translation model. If the source program is modified, the translation cost must be incurred again irrespective of the size of the modification. However, execution of the target program is efficient since the target program is in the machine language. Use of the interpretation model does not incur the translation overheads. This is advantageous if a program is modified between executions, as in program testing and debugging.

FUNDAMENTALS OF LANGUAGE PROCESSING

1. Lexical rules, which govern the formation of valid lexical units in the source language.

2. Syntax rules which govern the formation of valid statements in the source language.

3. Semantic rules which associate meaning with valid statements of the language.

The analysis phase uses each component of the source language specification to determine relevant information concerning a statement in the source program. Thus, analysis of a source statement consists of lexical, syntax and semantic analysis.

Lexical Analysis (Scanning)

Lexical analysis identifies the lexical units in a source statement. It then classifies the units into different lexical classes e.g. id's, constants etc. and enters them into different tables. This classification may be based on the nature of string or on the specification of the source language. (For example, while an integer constant is a string of digits with an optional sign, a reserved id is an id whose name matches one of the reserved names mentioned in the language specification.) Lexical analysis builds a descriptor, called a token, for each lexical unit. A token contain two fields— class code, and number in class, class code identifies the class to which a lexical unit belongs, number in class is the entry number of the lexical unit in the relevant table.

Syntax Analysis (Parsing)

Syntax analysis processes the string of tokens built by lexical analysis to determine the statement class, e.g. assignment statement, if statement, etc. It then builds an IC which represents the structure of the statement. The IC is passed to semantic analysis to determine the meaning of the statement.

Semantic analysis

Semantic analysis of declaration statements differs from the semantic analysis of imperative statements. The former results in addition of information to the symbol table, e.g. type, length and dimensionality of variables. The latter identifies the sequence of actions necessary to implement the meaning of a source statement. In both cases the structure of a source statement guides the application of the semantic rules. When semantic analysis determines the meaning of a sub tree in the IC. It adds information a table or adds an action to the sequence. It then modifies the IC to enable further semantic analysis. The analysis ends when the tree has been completely processed.

FUNDAMENTALS OF LANGUAGE SPECIFICATION

A specification of the source language forms the basis of source program analysis. In this section, we shall discuss important lexical, syntactic and semantic features of a programming language.

Programming Language Grammars

The lexical and syntactic features of a programming language are specified by its grammar. This section discusses key concepts and notions from formal language grammars. A language L can be considered to be a collection of valid sentences.

Each sentence can be looked upon as a sequence of words, and each word as a sequence of letters or graphic symbols acceptable in L. A language specified in this manner is known as a. formal language. A formal language grammar is a set of rules which precisely specify the sentences of L. It is clear that natural languages are not formal languages due to their rich vocabulary. However, PLs are formal languages.

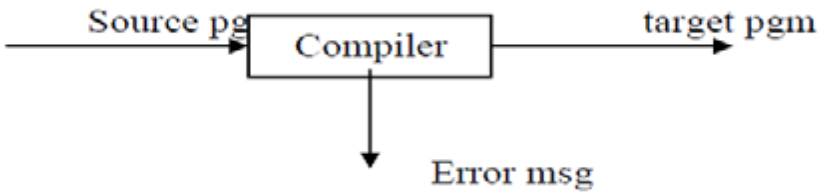
Terminal symbols, alphabet and strings

The alphabet of L, denoted by the Greek symbol Z, is the collection of symbols in its character set. We will use lower case letters a, b, c, etc. to denote symbols in Z.

A symbol in the alphabet is known as a terminal symbol (T) of L. The alphabet can be represented using the mathematical notation of a set, e.g. Σ {a, b, ...z, 0,1....9} Here the symbols {, ',' and} are part of the notation. We call them met symbols to differentiate them from terminal symbols. Throughout this discussion we assume that met symbols are distinct from the terminal symbols. If this is not the case, i.e. if a terminal symbol and a met symbol are identical, we enclose the terminal symbol in quotes to differentiate it from the Meta symbol. For example, the set of punctuation symbols of English can be defined as {:,;',’-,...} Where ',' denotes the terminal symbol 'comma'. A string is a finite sequence of symbols. We will represent strings by Greek symbols-α β γ, etc. Thus α = axy is a string over Σ. The length of a string is the Number of symbols in it. Note that the absence of any symbol is also a string, the null string. The concatenation operation combines two strings into a single string.

Compilers

Is a program that translate source code (high level language) to machine language(object code, target code, low level language)



A compiler is a program that translates a sentence from a source language (e.g. Java, Scheme, and LATEX) into a target language (e.g. JVM, Intel x86, PDF) while preserving its meaning in the process Compiler design has a long history (FORTRAN 1958)
We use natural language to communicate and we use programming languages to speak with computers.

Components of a Compiler:

a. Lexical Analysis
b. Syntax Analysis
c. Semantic Analysis
d. Intermediate code generation
e. Code Optimization
f. Code generation

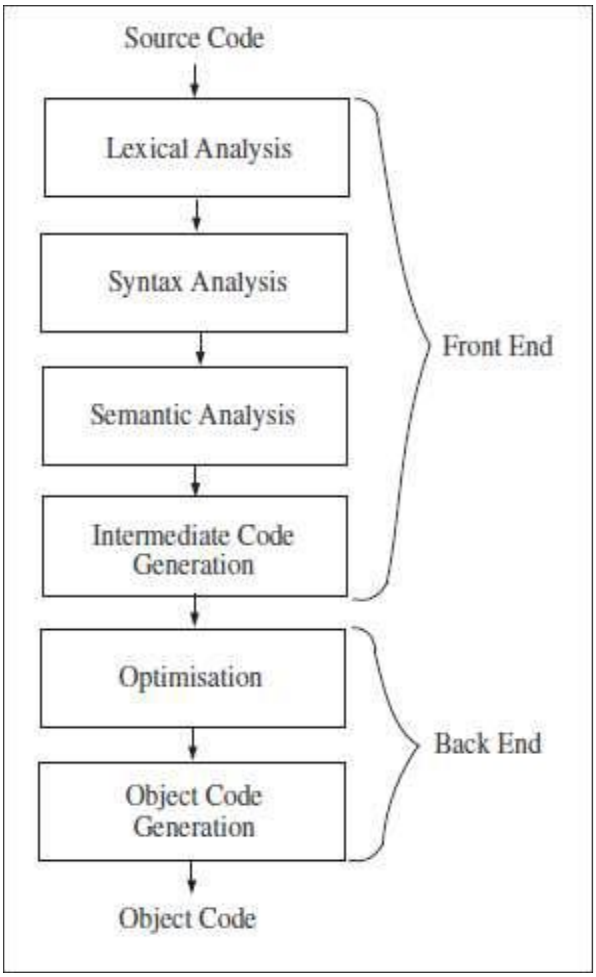


Figure 8: Conversion Process (High-level Language to Binary Language

The front end of a compiler translates a source program into an independent intermediate code, and then the back end of the compiler uses this intermediate code to generate the target code (which can be understood by the machine).

- Lexical Analysis: The first phase of a compiler which is also known as the scanner. It scans the source code as a stream of characters and converts it into meaningful lexemes (groups of logically related characters e.g. identifiers, punctuation, numbers, etc). The Lexical Analyzer breaks these lexemes into a series of tokens, by removing any whitespace or comments in the source code

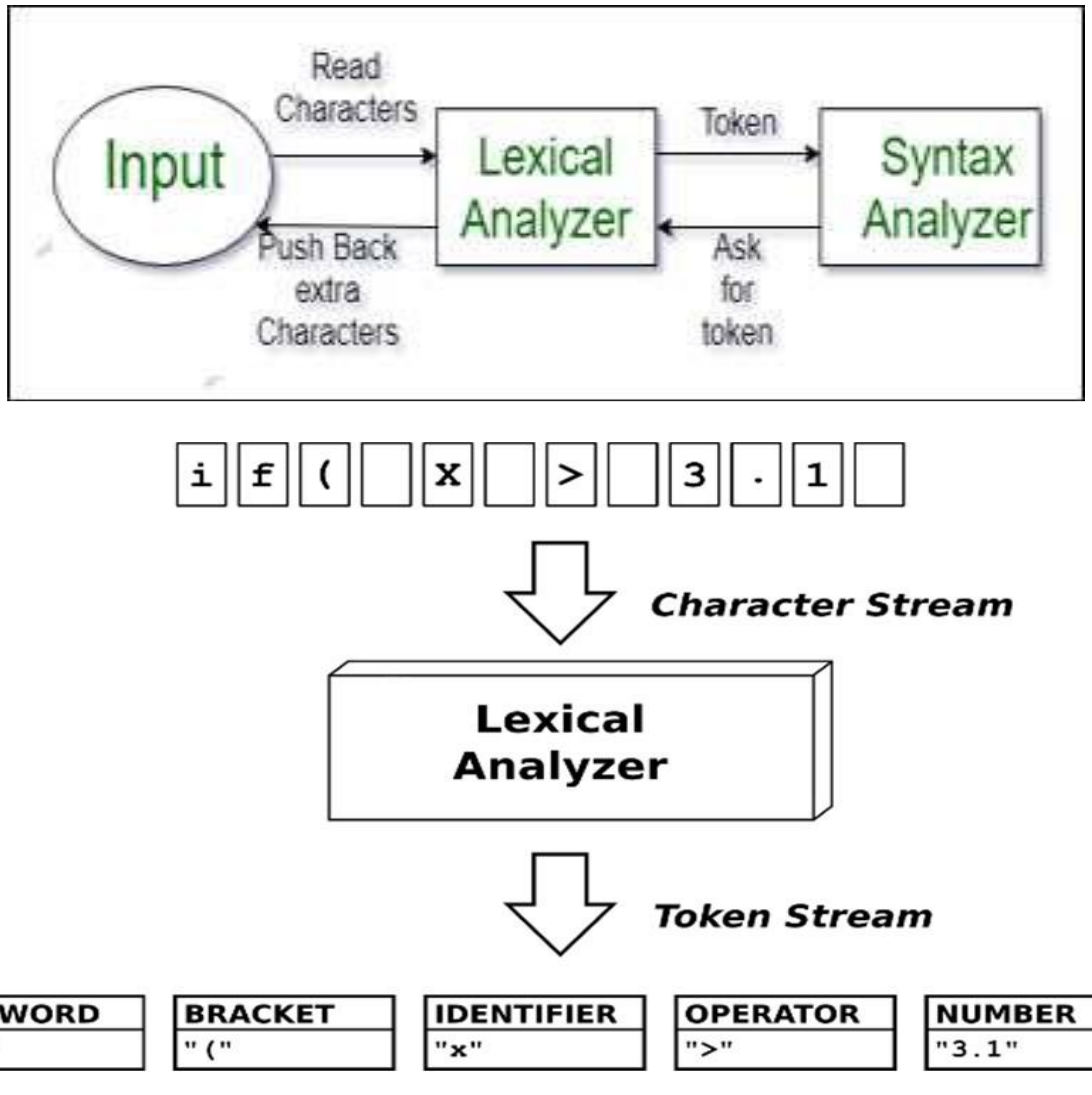


Figure 9: Role of lexical Analyzer

For example:

Input: x + 2 − y

Output: id(x), +, num (2), -, id(y)

The Scanner performs
a. Lexical Analysis
b. Collect character sequences into tokens. Example:
a [index] = 4+2, to convert this statement into token by the Scanner is:

Tokens:

a	identifier
[	left bracket
Index	identifier
]	right bracket
=	equal sign

4 number
+ plus sign
2 number

The scanner may also:

1. Enter identifiers into the symbol tables

2. Enter literals (numeric constants and strings) into the literals tables

Lexical Analysis (LA)

Maradona kicks the ball

Token Generation:

- **Maradona**
- **kicks**
- **the**
- **ball**

Who performs this ?

Lexical Analysis

X = Y + 30

Token Generation:

- **X** id_1
- **=** operator
- **Y** id_2
- **+** operator
- **30** literal/constant

What other functions Scanner can perform ?

- Syntax Analysis: The next phase is called the syntax analysis which can also be called hierarchical analysis or parsing. It takes the token produced by lexical analysis as input and groups the tokens of source Program into Grammatical Production. It generates a parse tree or syntax tree. Here, the parser checks if the expression made by the tokens is syntactically correct.

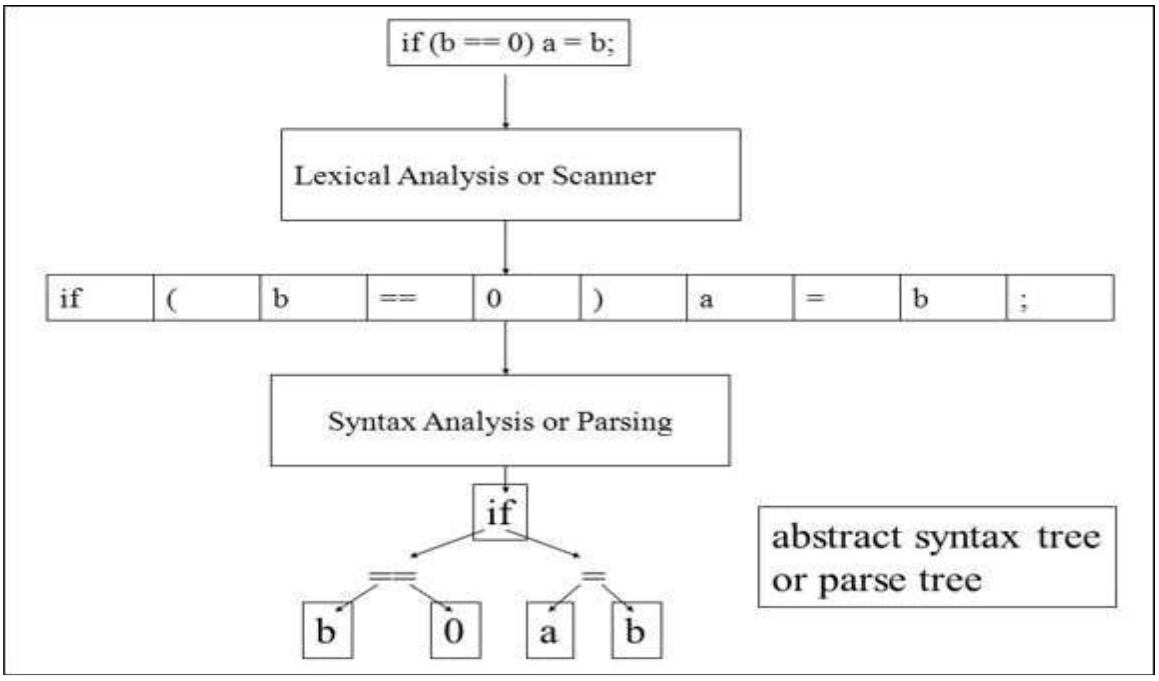
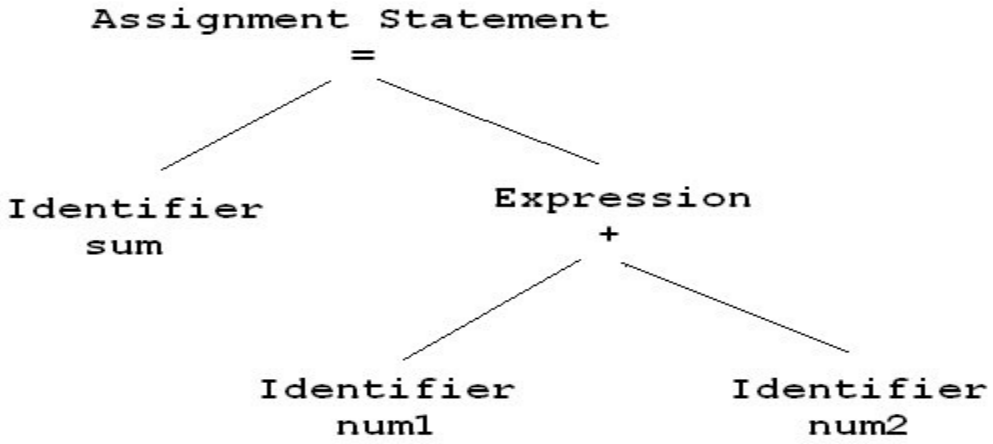


Figure 10: Generating Syntax Tree from Lexical Analyzer

Example:

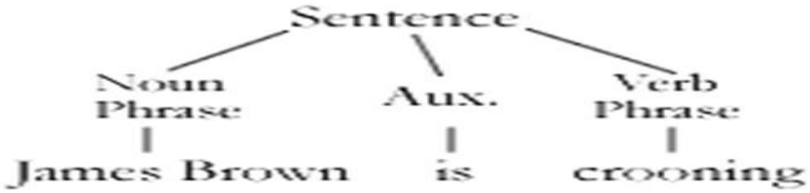
sum = num1 + num2

Consider the above programming statement. The Syntax Analyzer will create a parse tree from the tokens

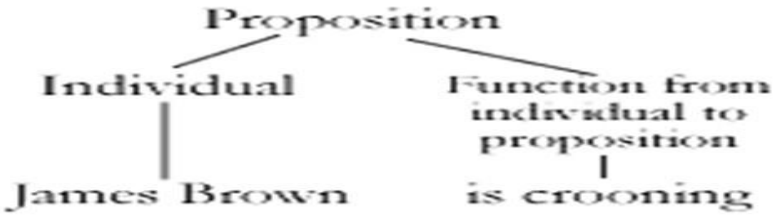


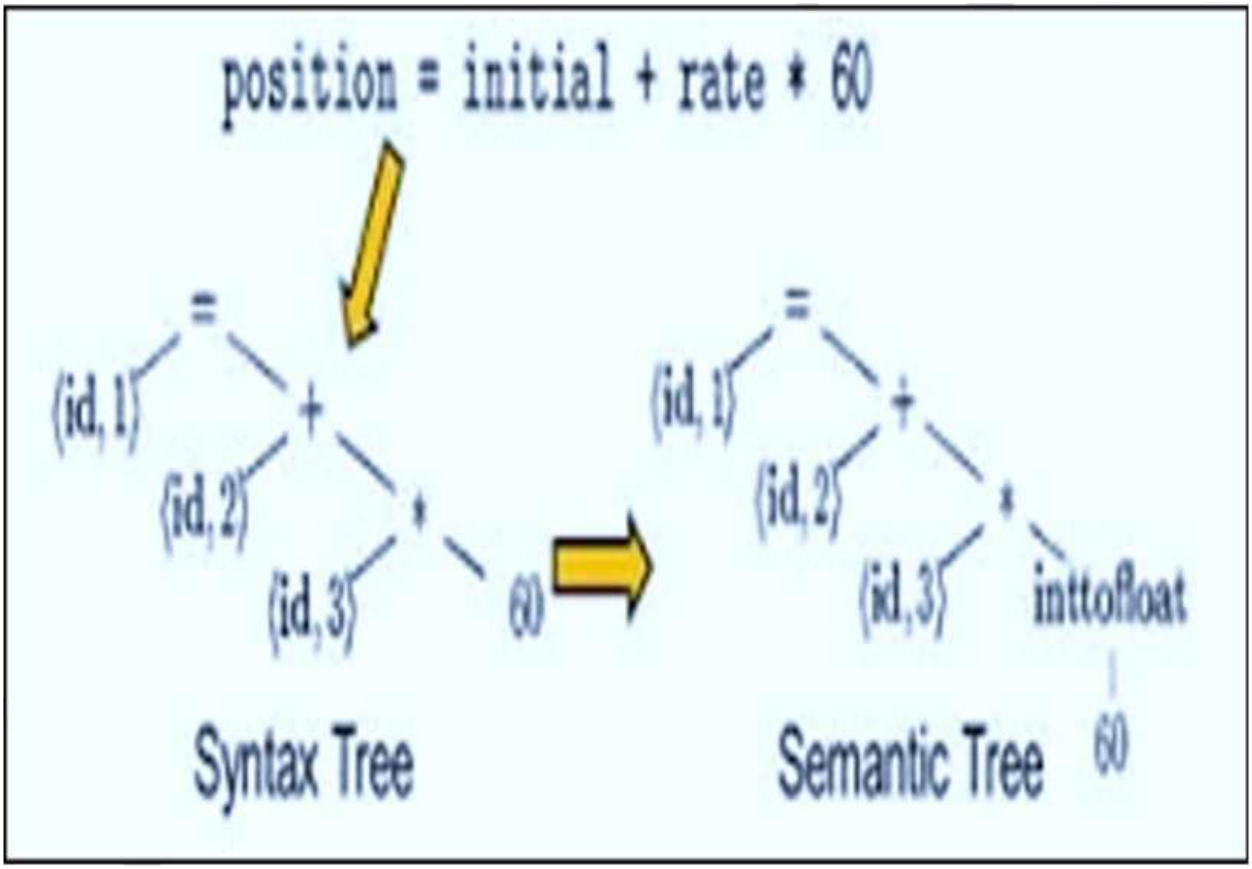
- Semantic Analysis: This phase checks the source program for semantic errors, and gathers type information for the intermediate code generation phase. It checks whether the parse tree constructed follows the rules of language and ensuring that the declarations and statements of a program are semantically correct. For example, assignment of values is between compatible data types, and adding string to an integer Also, the semantic analyzer keeps track of identifiers, their types and expressions; whether identifiers are declared before use or not etc. The semantic analyzer produces an annotated syntax tree as an output.

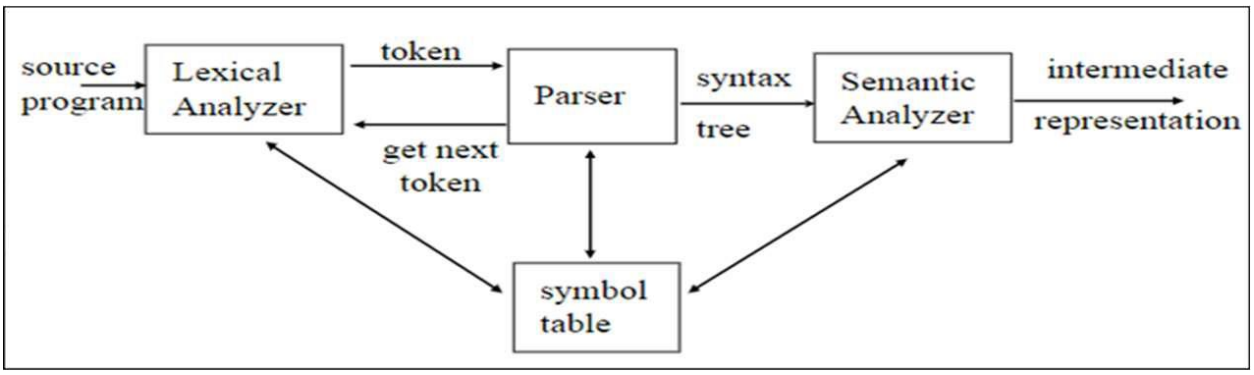
Example of a syntactic tree



Example of (one kind of) semantic tree





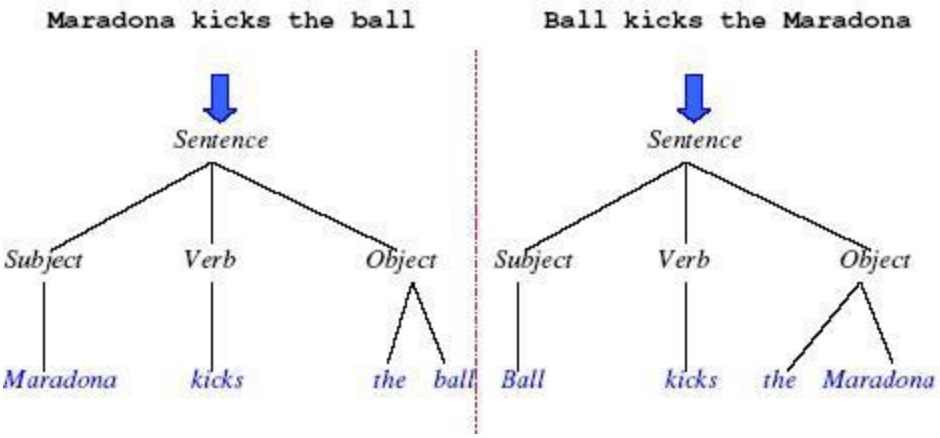


The parser performs:

a. Syntax Analysis
b. Builds a parse tree or syntax tree

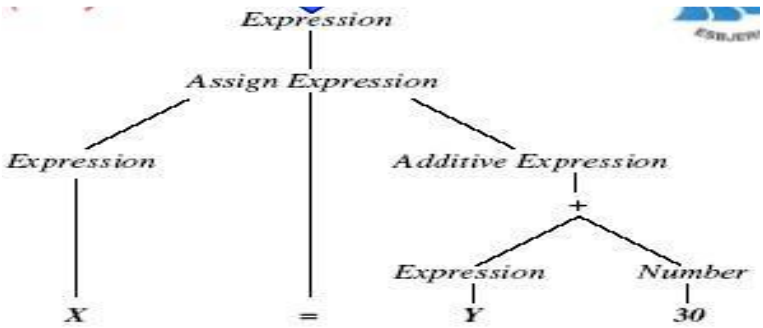
Parse tree for a [index] = 4 + 2

Structure of the program is determined by SA. Some thing similar to grammatical analysis.

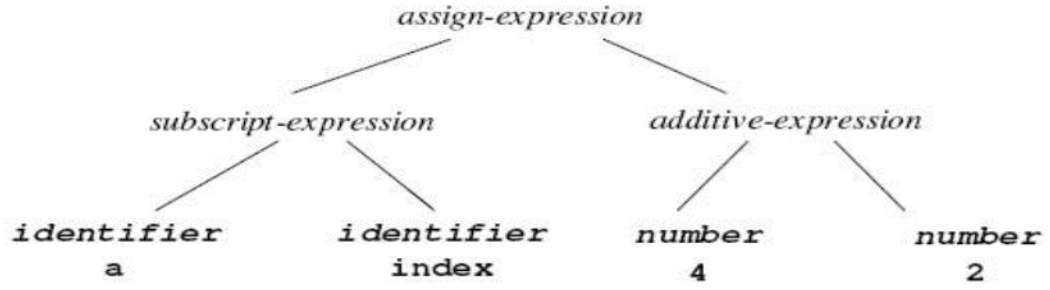


An abstract syntax tree s a more concise version of a parse tree

X = y + 30

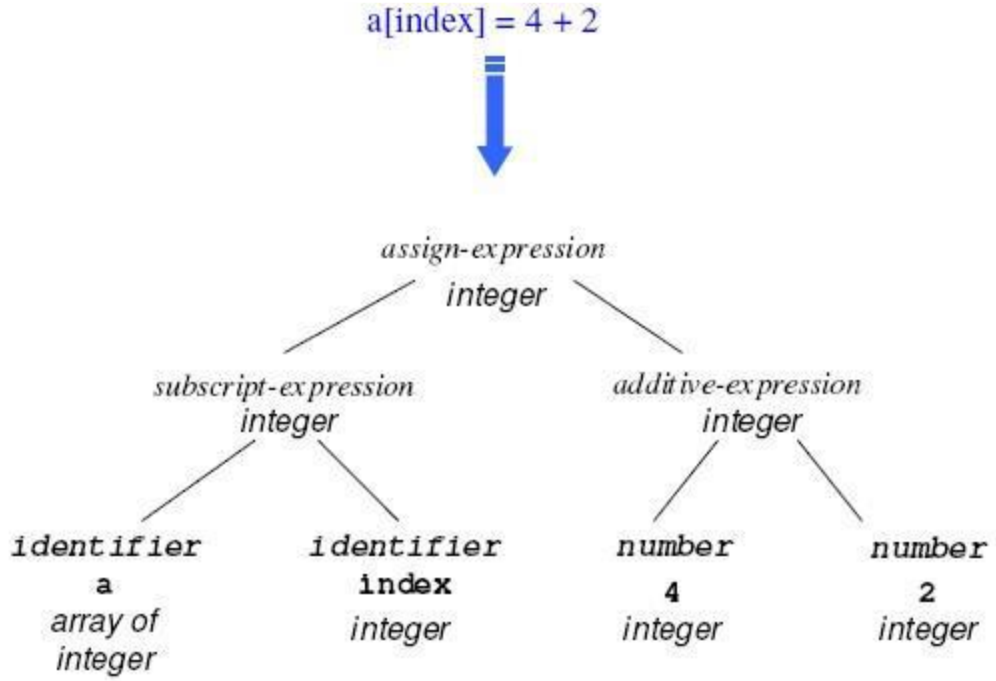


Sometime syntax tree is also called as Abstract Syntax tree and could be a 'trimmed' version of the parse tree with only essential Information for Example : a [index] = 4+ 2



Semantic analyser

This attach meaning of token; For example to the same expression



- Intermediate Code Generation: In computing, code generation is the process by which a compiler's code generator converts some intermediate representation of source code into a form that can be readily executed by a machine. After semantic analysis the compiler generates an intermediate code of the source code for the target machine. It represents a program for some abstract machine. It is in between the high-level language and the machine language. This intermediate code should be generated in such a way that it makes it easier to be translated into the target machine code

One of the popular intermediate codes is three—addresses code. A three address code:

- Each statement contains at most 3 operands; in addition to":=",

Another example: X = Y + 30

$$\begin{aligned} Temp1 &= 30 \\ Temp2 &= Y \\ Temp3 &= Temp2 + Temp1 \\ X &= Temp3 \end{aligned}$$

- Code Generation: In this phase, the code generator takes the optimized representation of the intermediate code and maps it to the target machine language. The code generator translates intermediate code into object code, allocating memory allocations for data, and selecting registers. Sequence of instructions of machine code performs the task as the intermediate code would do.

Example: X= Y + 30
MOV Y, R1
ADD 30, R1
MOV R1, X

ALGORITHMICS TOOLS

I. Token : Using regular expression
II. Scanner :Implementation of finite state machine to recognize tokens
III. Parser : An automation (that is uses stack), based on grammar rules in standard format (Back Nur form BNF)
IV. Semantic Analyser and Code Generator : Recursive evaluators based on semantic rules for attributes (properties of language constructs)

Error Handling in Compiler

An error is the blank entries in the symbol table. It is one of the difficult parts of a compiler design. The tasks of the error handling process are;

- Detect each error
- Handle a wide range or multiple errors
- Report the error to the user
- Make some recovery strategy and implement them to handle the error.
- During the whole process, the processing time of program should not be slow.

There are basically two types of error:

i. A Run Time Error: This is an error that takes place during execution of a program, and it usually happens because of invalid input data. The lack of sufficient memory to run an application or a memory conflict with another program and logical error are example of this. Logic errors occur when executed code does not produce the expected result. These are handled by thorough program debugging.
ii. Compile-time Error: It happens at compile time, before the program execution. Syntax error or missing file reference that prevents the program from successfully compiling is the best example here.

Kind of errors

Syntax error

If (x = = 0) y + = z + r;}

Semantic

Int x = "Hello, world! " ;

Runtime:

int x = 2 ;

…

Double y = 3.142 / (x - 2) ;

- A compiler must handle syntax and semantic errors, but not runtime error(whether a runtie will occur is million dollar question)
- Someties a compiler is require to generate code to catch runtime errors an handle them in some graceful way(either with or without exception handling).This, too,is often difficult

Major Compiler Data Structure

A. Tokens
   i. Represented as an enumerated type
   ii. May require other information :
      a. Spelling of identifier
      b. Numeric Value
   iii. Scanner needs generated only one token at a time (Single symbol lookaead)

B. Syntax Tree
   i. A link structure built by the parser, with information added by the semantic analyser.
   ii. Each node is a record whose fields contain information about the syntactic construct which the node represents

iii. Node may be represented using a variant record

C. Symbol Table

Symbol table is an important data structure created and maintained by compilers in order to store information about the occurrence of various entities such as variable names, function names, objects, classes, interfaces, etc. Symbol table is used by both the analysis and the synthesis parts of a compiler.

1. Keeps informmation about identifiers

2. Efficient insertion an lookup is required ie hash table or tree structure may be used

3. Several tables may be maitaine in a list or stack

Interpreter

A Computer language processor that translates a program line-by-line (statement-by-statement) and carries out the specified actions in sequence. An assembler or compiler completely translates a program written in a high-level language (the source program) into a machine-language program (the object program) for later execution. Whereas a compiled-program executes much faster than an interpreted-program, an interpreter allows examination and modification of the program while it is running (executing). The difference lies in the way they read the source code or input. A compiler reads the whole source code at once, creates tokens, checks semantics, generates intermediate code, executes the whole program and may involve many passes. In contrast, an interpreter reads a statement from the input, converts it to an intermediate code, executes it, then takes the next statement in sequence. If an error occurs, an interpreter stops execution and reports it. Whereas a compiler reads the whole program even if it encounters several error

1. Stores constants and strings used in a program
2. Data in literal table applies globally to a program → deletions are not necessary

e. Intermediate code
   1. Could be kept in an array, temporary file, or linked list
   2. Representations include P-code and 3-address code

f. Temporary files
   1. May not be used if memory constraints are not a problem
   2. **Backpatching** of addresses necessary during translation

INTRODUCTION TO ASSEMLER AND ASSEMBLY LANGUAE

Encoding instruction as binary numbers is natural and efficient for computers. Human, however, have a great deal of difficulty understanding and manipulating these numbers. People read and write symbols (words) much better than long sequences of digits

What is an assembler?

A tool called an **assembler** translates assembly language into binary instructions. Assemblers provide a friendlier representation than a computer's 0s and 1s that simplifies writing and reading programs. Symbolic names for operations and locations are one facet of this representation. Another facet is programming facilities that increase a program's clarity.

An assembler reads a single assembly language **source file** and produces an **object file** containing machine instructions and bookkeeping information that helps combine several object files into a program. Figure (11) illustrates how a program is built. Most programs consist of several files—also called **modules**— that are written, compiled, and assembled independently. A program may also use prewritten routines supplied in a **program library**. A module typically contains References to subroutines and data defined in other modules and in libraries. The code in a module cannot be executed when it contains unresolved References to labels in other object files or

libraries. Another tool, called a linker, combines a collection of object and library files into an executable file, which a computer can run.

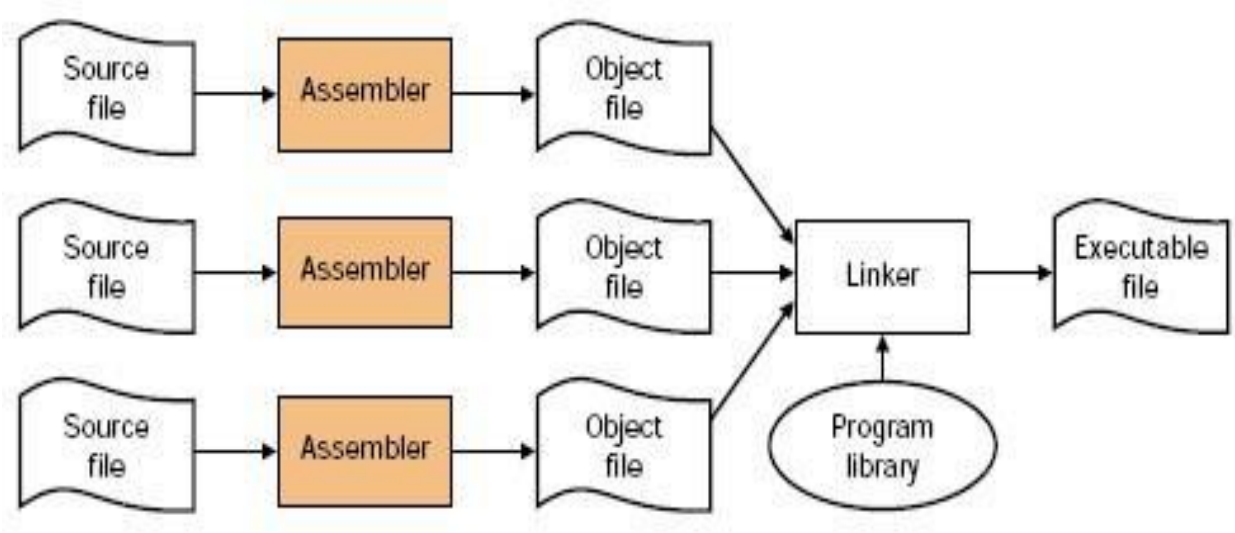


FIGURE 11: The process that produces an executable file. An assembler translates a file of assembly language into an object file, which is linked with other files and libraries into an executable file.

1) Assembler = a program to handle all the tedious mechanical translations

2) Allows you to use:

• Symbolic op codes

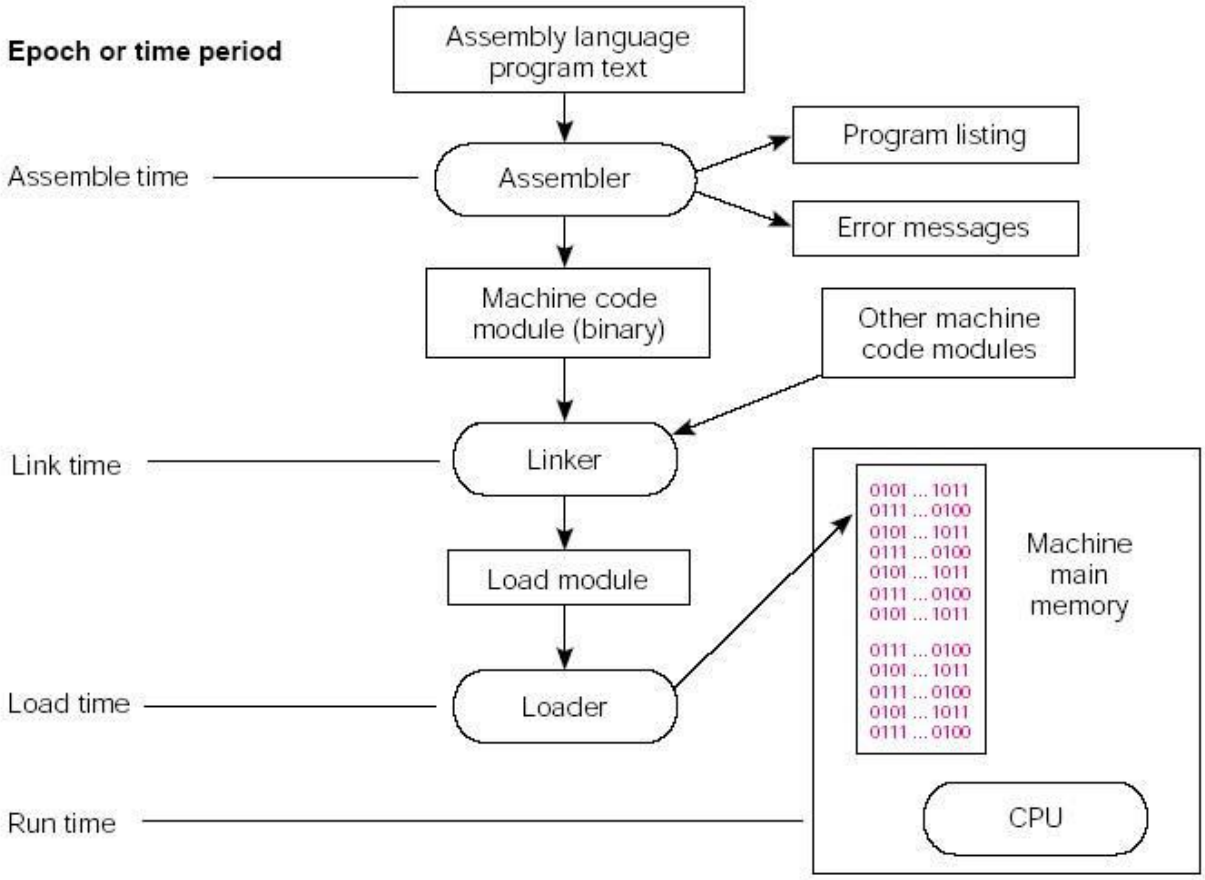
• Symbolic operand values

• Symbolic addresses

3) The Assembler

- keeps track of the numerical values of all symbols
- translates symbolic values into numerical values

4) Time Periods of the Various Processes in Program Development



5) The Assembler Provides:

a. Access to all the machine's resources by the assembled program. This includes access to the entire instruction set of the machine.

b. A means for specifying run-time locations of program and data in memory.

c. Provide symbolic labels for the representation of constants and addresses.

d. Perform assemble-time arithmetic.

e. Provide for the use of any synthetic instructions.

f. Emit machine code in a form that can be loaded and executed.

g. Report syntax errors and provide program listings

h. Provide an interface to the module linkers and program loader.

i. Expand programmer defined macro routines

Disadvantages of Assembly

- Programmer must manage movement of data items between memory locations and the ALU.
- Programmer must take a "microscopic" view of a task, breaking it down to manipulate individual memory locations.
- Assembly language is machine-specific.
- Statements are not English-like (Pseudo-code)

Linker & Loader

LINKER

In high level languages, some built in header files or libraries are stored. These libraries are predefined and these contain basic functions which are essential for executing the program. These functions are linked to the libraries by a program called Linker.

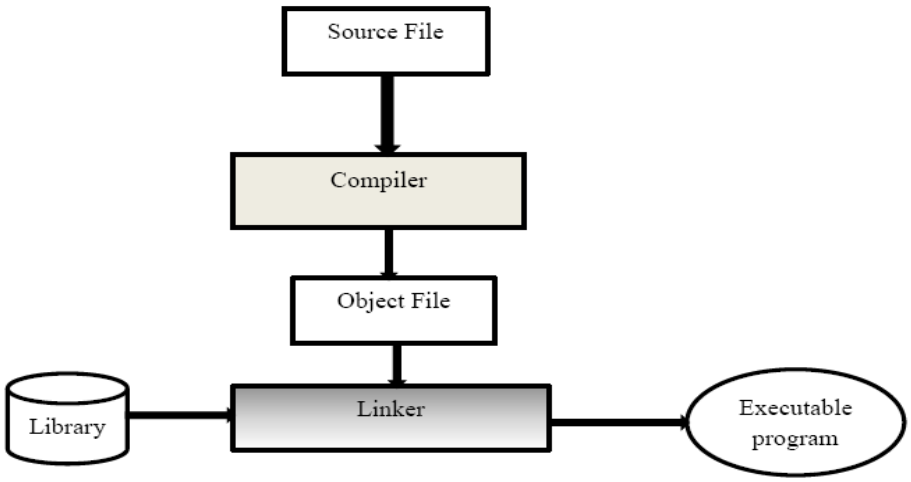


Figure 12: Compilation Process

A software processor, which performs some low level processing of the programs input to it, produces a ready to execute program form. The basic loading function is that of locating a program in an appropriate area of the main store of a computer when it is to be executed. A loader often performs the two other important functions.

The loader, which accepts the program form, produced by a translator & certain other program forms from a library to produce one ready – to – execute machine language program. A unit of input to the loader is known as an object program or an object module. The process of merging many object modules to from a single machine language program is known as linking. The function to be performed by: Assigning of loads the storage area to a program. Loading of a program into the assigned area. Relocation of a program to execute properly from its load time. Loader, linking loaders, linkage editors are used in software literature

LOADER:

The loader is program, which accepts the object program decks, prepares this program for execution by the computer and initializes the execution. In particular the loader must perform four functions:

- Allocate space in memory for the program (allocation).
- Resolve symbolic references between objects decks (linking).
- Adjust all address dependent locations, such as address constants, to correspond to the allocated space (relocation).
- Physically place the machine instructions and data into memory (loading).

SYSTEM CONTROL SOFTWARE (OPERATING SYSTEM OS)

What is an Operating System?

An **Operating System (OS)** is software that acts as an interface/ intermediary between computer hardware components and the user. Every computer system must have at least one operating system to run other programs. Applications like Browsers, MS Office, Notepad Games, etc., need some environment to run and perform its tasks.

The OS helps you to communicate with the computer without knowing how to speak the computer's language. It is not possible for the user to use any computer or mobile device without having an operating system.

Brief History of OS

- Operating systems were first developed in the late 1950s to manage tape storage
- The General Motors Research Lab implemented the first OS in the early 1950s for their IBM 701
- In the mid-1960s, operating systems started to use disks
- In the late 1960s, the first version of the Unix OS was developed
- The first OS built by Microsoft was DOS. It was built in 1981 by purchasing the 86-DOS software from a Seattle company
- The present-day popular OS Windows first came to existence in 1985 when a GUI was created and paired with MS-DOS.

Operating System in computer science is the basic software that controls a computer. It controls the execution of application programs and acts as an interface between the user applications and computer hardware. The purpose of an OS is to provide a platform on which a user can execute programs in a convenient and efficient manner. Not all computers have operating systems. The computer that controls the microwave oven in your kitchen, for example, doesn't need an operating system. It has one set of tasks to perform, very straightforward input to expect (a numbered keypad and a few pre-set buttons) and simple, never-changing hardware to control.

Hence, a computer in a microwave oven simply runs a single hard-wired program all the time.

All desktop computers have operating systems. The most common are the Windows family of operating systems developed by Microsoft, the Macintosh operating systems developed by Apple and the UNIX family of operating systems (which have been developed by a whole history of

individuals, corporations and collaborators). In any device that has an operating system, there's usually a way to make changes to how the device works. One of the reasons operating systems are made out of portable code rather than permanent physical circuits is so that they can be changed or modified without having to scrap the whole device. For a desktop computer user, this means you can add a new security update, system patch, new application or even an entirely new operating system rather than junk your computer and start again with a new one when you need to make a change.

How an OS Works

However, when you turn on the power to a computer, the first program that runs is usually a set of instructions kept in the computer's read-only memory (ROM). This code examines the system hardware to make sure everything is functioning properly. The power-on self-tests (POST) checks the CPU, memory, and basic input-output systems (BIOS) for errors and stores the result in a special memory location. Once the POST has successfully completed, the software loaded in ROM (sometimes called the BIOS or firmware) will begin to activate the computer's disk drives. In most modern computers, when the computer activates the hard disk drive, it finds the first piece of the operating system: the bootstrap loader. The bootstrap loader is a small program that has a single function: It loads the operating system into memory and allows it to begin operation. In the most basic form, the bootstrap loader sets up the small driver programs and controls the various hardware subsystems of the computer. It sets up the divisions of memory that hold the operating system, user information and applications. Then it turns control of the computer over to the operating system.

Moreover, OS are either single-tasking or multitasking. The more primitive single-tasking operating systems can run only one process at a time. For instance, when the computer is printing a document, it cannot start another process or respond to new commands until the printing is completed. All modern operating systems are multitasking and can run several processes simultaneously. In most computers, there is only one central processing unit (CPU), so a multitasking OS creates the illusion of several processes running simultaneously on the CPU. The most common mechanism used to create this illusion is time-slice multitasking, whereby each process is run individually for a fixed period of time. If the process is not completed within the allotted time, it is suspended and another process run. This exchanging of processes is called context switching. The OS performs the “bookkeeping” that preserves a suspended process. It also has a mechanism, called a scheduler that determines which process will be run next. The scheduler runs short processes quickly to minimize detectable delay. The processes appear to run simultaneously because the user's sense of time is much slower than the processing speed of the computer.

User Interface Functions:

Its purpose is to provide the use of OS resources for processing a user’s computational requirements. OS user interfaces use command languages. For this, the user uses Command to set up an appropriate computational structure to fulfill his computational requirements.

An OS can define a variety of computational structures. A sample list of computational structures is as follows:

1. A single program

2. A sequence of single program

3. A collection of programs

1. The single program consist the execution of a program on a given set of data. The user initiates execution of the program through a command. Two kinds of program can exist – Sequential and concurrent.
2. A sequential program is the simplest computational structure.
3. In concurrent program the OS has to be aware of the identities of the different parts, which can execute concurrently.

Operating System Services

An operating system provides services to programs and to the users of those programs. It provides an environment for the execution of programs. Operating system makes the programming task easier. The services provided by one operating system are different than other operating system. The common services are listed below.

1. Program Execution: Operating system loads a program into memory and executes the program. The program must be able to end its execution, either normally or abnormally.

2. Input/output Operation: Program may require any I/O device while running. So, operating system must provide the required I/O.

3. File System Manipulation: Program needs to read a file or write a file. The operating system gives the permission to the program for operation on file.

4. Communications: Data transfer between two processes is required for some time. The both Processes are on the one computer or on different computer but connected through computer Network. Communication may be implemented by two methods: shared memory and message Passing

5. Error detection: Error may occur in CPU, in I/O devices or in the memory hardware. The operating system constantly needs to be aware of possible errors. It should take the appropriate action to ensure correct and consistent computing. Operating system with multiple users provides following services Resource allocation, Accounting and Protection.

An operating system is a lower level of software that user programs run on. OS is built directly on the hardware interface and provides an interface between the hardware and the user program. It shares characteristics 'with both software and hardware.

We can view an operating system as a resource allocator. OS keeps track of the status of each resource and decides who gets a resource, for how long, and when and makes sure that different programs and users running at the same time but do not interfere with each other. It is also responsible for security, ensuring that unauthorized users do not access the system. The primary objective of operating systems is to increase productivity of a processing resource, such as

computer hardware or users. The operating system is the first program on a computer when the computer boots up.

1 Resource Allocation: If there are more than one user or jobs running at the same time, then resources must be allocated to each of them. Operating system manages different types of resources. Some resources require special allocation code, i.e., main memory, CPU cycles and file storage.

2 Accounting: Logs of each user must be kept. It is also necessary to keep record of which user uses how much and what kinds of computer resources. This log is used for accounting purposes. The accounting data may be used for statistics or for the billing. It also used to improve system efficiency.

3 Protection: Protection involves ensuring that all access to system resources is controlled. Security starts with each user having to authenticate to the system, usually by means of a password. External I/O devices must be also protected from invalid access attempts. In protection, all the access to the resources is controlled. In multiprocessor environment, it is possible that, one process to interface with the other, or with the operating system, so protection is required.

Functions of an OS

Operating System is responsible for functioning of the computer system. To do that it carries out these three broad categories of activities;

- Essential functions − Ensures optimum and effective utilization of resources
- Monitoring functions − Monitors and collects information related to system performance
- Service functions − Provides services to users

Let us look at some of the most important functions associated with these activities

- Processor Management: The term process refers to an executing set of machine
- Instructions. Program by itself is not a process. A program is a passive entity. The operating system responds by creating a process. A process needs certain resources, such as CPU time, memory, files and I/O devices. These resources are either given to the process when it is created or allocated to it while it is running. When the process terminates, the operating system will reclaim any reusable resources.

However, managing a computer's CPU to ensure its optimum utilization is called Processor Management. Managing processor basically involves allocating processor time to the tasks that need to be completed. This is called job scheduling. Jobs must be scheduled in such a way that

- there is maximum utilization of CPU,
- turnaround time (i.e., minimum time required to complete each job),
- minimum waiting time,
- Fastest possible response time for each job, and a maximum throughput (where throughput is the average time taken to complete each task).

We have two methods of job scheduling done by an OS i.e. Preemptive scheduling and Non-Preemptive scheduling

Preemptive Scheduling: In this type of scheduling, next job to be done by the processor can be scheduled before the current job completes. If a job of higher priority comes up, the processor can be forced to release the current job and take up the next job.

Non-Preemptive scheduling: In this type of scheduling, job scheduling decisions are taken only after the current job completes. A job is never interrupted to give precedence to higher priority jobs.

- Memory Management: Memory space is very important in modern computing environment, so memory management is an important role of operating systems. The process of regulating computer memory and using optimization techniques to enhance overall system performance is called memory management.

Computers have two types of memory; primary and secondary. The primary or main memory is a fast but expensive storage that can be accessed directly by the CPU. While secondary memory is cheap but slower. For any program to be executed, it should be first loaded in the main memory. Once a program request is accepted, OS allocates it primary and secondary storage areas as per requirement. Once execution is completed, the memory space allocated to it is freed. OS uses many storage management techniques to keep a track of all storage spaces that are allocated or free.

Contiguous Storage Allocation: This is the simplest technique where contiguous memory locations are assigned to each program. OS has to estimate the amount of memory required for the complete process before allocation.

Non-contiguous Storage Allocation: As the name suggests, program and associated data need not be stored in contiguous locations. The program is divided into smaller components and each component is stored in a separate location. A table keeps a record of where each component of the program is stored. When the processor needs to access any component, OS provides access using this allocation table.

- Device Management: OS keeps tracks of all devices connected to system via their respective drivers. The process of implementation, operation and maintenance of a device by operating system is called Device Management. OS uses utility software called device driver as interface to the device. When many processes access the devices or request access to the devices, the OS manages the devices in a way that efficiently shares the devices among all processes. Processes

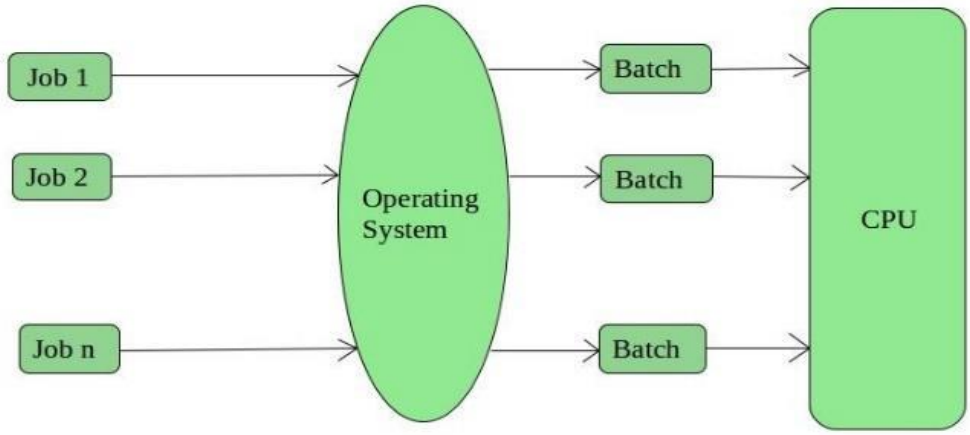
Access devices through system call interface (provide the interface between a process and the OS). A system call instruction is an instruction that generates an interrupt which causes the OS to gain control of the processor.

- File Management: A file consists of a sequence of bits, bytes, lines or records whose meanings are defined by their creators. Data and information is stored on computers in form of files. Managing file system to enable users to keep their data safely and correctly is an important function of OS. Managing file systems by OS is called File Management. The operating system is responsible for the following in connection with file management.
- Creating new files for storing data
- Deleting

- Sharing
- Securing data through passwords and encryption
- Updating
- Recovery in case of system failure
- **Types of Operating System (OS)**
- As computers and computing technologies have evolved over the years, operating system have kept evolving with time to accommodate more and more sophisticated tasks. An Operating System performs all the basic tasks like managing files, processes, and memory. Thus operating system acts as the manager of all the resources, i.e. resource manager. Thus, the operating system becomes an interface between user and machine. Here we discuss some of the most common types of operating systems in use today.
  Following are the popular types of Operating System:
- Batch Operating System
- Multitasking/Time Sharing OS
- Multiprocessing OS
- Real Time OS
- Distributed OS
- Network OS
- Mobile OS

❖ **Batch operating system**

Batch Operating System: This is an old processing technique and rarely used at present. In this operating system, the user will prepare the job on an off-line device like punch cards and hand it to a computer operator. The computer operator would then group the jobs according to their computing requirements and execute them in batches to ensure a faster processing speed.



Advantages of Batch Operating System:

 Processors of the batch systems know how long the job would be when it is in queue

 Multiple users can share the batch systems

 It is easy to manage large work repeatedly in batch systems

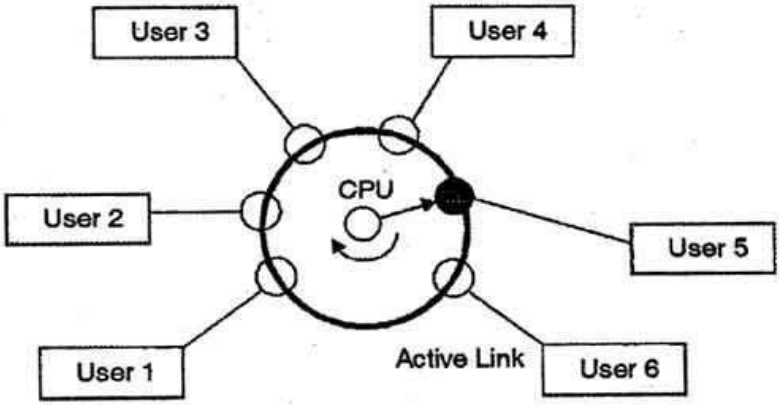
Disadvantages of Batch Operating System:

- The computer operators should be well known with batch systems
- Batch systems are hard to debug
- It is sometimes costly
- The other jobs will have to wait for an unknown time if any job fails
- Lack of interaction between the user and the job.
- CPU is often idle, because the speed of the mechanical I/O devices is slower than the CPU.
- Difficult to provide the desired priority.

Multi-Tasking/Time-sharing Operating systems

Time-sharing operating system enables people located at a different terminal (shell) to use a single computer system at the same time. The processor time (CPU) which is shared among multiple users is termed as time sharing. In Time-Sharing Systems, the objective is to minimize response time.

Multiple jobs are executed by the CPU by switching between them, but the switches occur so frequently. Thus, the user can receive an immediate response. For example, in a transaction processing, the processor executes each user program in a short burst or quantum of computation. That is, if **n** users are present, then each user can get a time quantum. When the user submits the command, the response time is in few seconds at most. The operating system uses CPU scheduling and multiprogramming to provide each user with a small portion of a time. Computer systems that were designed primarily as batch systems have been modified to time-sharing systems.



Advantages of Timesharing operating systems are as follows −

 provides the advantage of quick response.

 Avoids duplication of software.

 Reduces CPU idle time.

Disadvantages of Time-sharing operating systems are as follows −

 Problem of reliability.

 Question of security and integrity of user programs and data.

 Problem of data communication.

Real time OS

A real-time system is defined as a data processing system in which the time interval required to process and respond to inputs is so small that it controls the environment. The time taken by the system to respond to an input and display of required updated information is termed as the **response time**. So, in this method, the response time is very less as compared to online processing. Real-time systems are used when there are rigid time requirements on the operation of a processor, or the flow of data and real-time systems can be used as a control device in a dedicated application. A real-time operating system must have well-defined, fixed time constraints, otherwise the system will fail. For example, scientific experiments, medical imaging systems, industrial control systems, weapon systems, robots, air traffic control systems, etc.

There are two types of real-time operating systems.

 Hard-real-time systems: Hard real-time systems guarantee that critical tasks complete on time. In hard real-time systems, secondary storage is limited or missing, and the data is stored in ROM. In these systems, virtual memory is almost never found.

 Soft real-time systems: Soft real-time systems are less restrictive. A critical real-time task gets priority over other tasks and retains the priority until it completes. Soft real-time systems have limited utility than hard real-time systems. For example, multimedia, virtual reality, Advanced Scientific Projects like undersea exploration and planetary rovers, etc.

Comparison between Hard and Soft Real Time System

• Hard real time system guarantees that critical tasks complete on time. To achieve this, all delays in the system must be bounded i.e. the retrieval of stored data to the time that it takes the operating system to finish any request made of it. Soft real time system are less restrictive than the hard real time system. In soft real time, a critical real time task gets priority over other tasks and retains that priority until it complete.

Time constraints are the main properties for the hard real time systems. Since none of the operating system support hard real time system, Kernal delays need to be bounded in soft real time system. Soft real time systems are useful in the area of multimedia, virtual reality and advance scientific projects. Soft real time systems cannot be used in -robotics and industrial control because of their lack of deadline support. Soft real time system requires two conditions to implement. CPU scheduling must be priority based and dispatch latency must be small.

• **Handheld System:**

• Personal Digital Assistants (PDA) is one type of handheld systems. Developing such device is the complex job and many challenges will face by developers. Size of these system is small i.e. height is 5 inches and width is 3 inches.

• Due to the limited size, most handheld devices have a small amount of memory, include slow processors and small display screen. Memory of handheld system is in the range of 512 kB to 8

MB. Operating system and applications must manage memory efficiently. This includes returning all allocated memory back to the memory manager once the memory is no longer needed. Developers are working only on confines of limited physical memory because any handheld devices not using virtual memory.

• Speed of the handheld system is major factor. Faster processors require for handheld systems. Processors for most handheld devices often run at a fraction of the speed of a processor in a Pc. Faster processors require more power. Larger battery requires for faster processors.

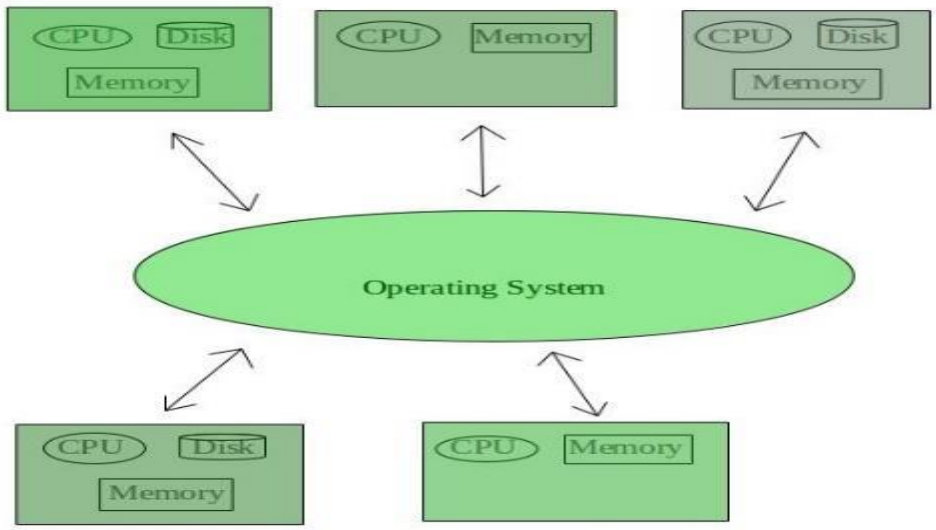
• For mimimum size of handheld devices, smaller, slower processors which consumes less power are used. Typically small display screen is available in these devices. Display size of handheld device is not more than 3 inches square.

• At the same time, display size of monitor is up to 21 inches. But these handheld device provides the facility for reading email, browsing web pages on smaller display. Web clipping is used for displaying web page on the handheld devices.

• Wireless technology is also used in handheld devices. Bluetooth protocol is used for remote access to email and web browsing. Cellular telephones with connectivity to the Internet fall into this category.

Distributed Operating System

Distributed systems use multiple processors located in different machines to provide very fast computation to real time applications and multiple users. The processors communicate with one another through various communication lines (such as high-speed buses or telephone lines). These are referred as **loosely coupled systems** or distributed systems. Processors in a distributed system may vary in size and function. These processors are referred as sites, nodes, computers, and so on.



The advantages of distributed systems are as follows −

 with resource sharing facility, a user at one site may be able to use the resources available at another. Speedup the exchange of data with one another via electronic mail.

 If one site fails in a distributed system, the remaining sites can potentially continue operating.

 Better service to the customers.

 Reduction of the load on the host computer.

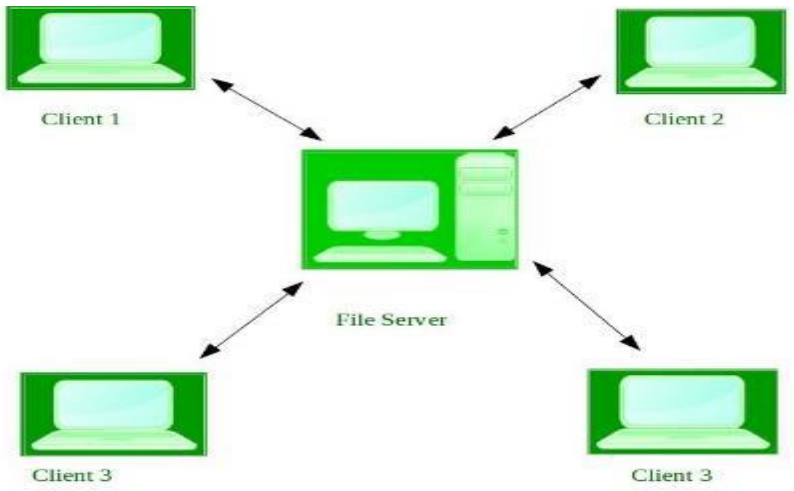
 Reduction of delays in data processing.

Network Operating System

Network Operating System runs on a server. It provides the server the capability to manage data, users, groups, security, application, and other networking functions.

The primary purpose of the network operating system is to allow shared file and printer access among multiple computers in a network, typically a local area network (LAN), a private network or to other networks.

Examples of network operating systems include Microsoft Windows Server 2003, Microsoft Windows Server 2008, UNIX, Linux, Mac OS X, Novell NetWare, and BSD.



The advantages of network operating systems are as follows −

 Centralized servers are highly stable.

 Security is server managed.

 Upgrades to new technologies and hardware can be easily integrated into the system.

 Remote access to servers is possible from different locations and types of systems.

The disadvantages of network operating systems are as follows −

 High cost of buying and running a server.

 Dependency on a central location for most operations.

Regular maintenance and updates are required.

Mobile OS

Mobile operating systems are those OS which is especially that are designed to power smartphones, tablets, and wearables devices. Some most famous mobile operating systems are Android and iOS, but others include BlackBerry, Web, and watch OS.

Classifications of Operating System

The classification of an operating system is a grouping that differentiates or identifies the operating system based on how it works the type of hardware it controls and the applications it supports.

i Single-user versus Multi-user OS: Single user type of operating system has to deal with one person at a time. An example of this type of operating system can be found on mobile phones. There can be only one user using that mobile. In a multiuser OS, more than one user can use the same system at the same time through the multi I/O terminal or through the network. For example: windows, Linux, Mac.

ii Single-processor versus Multi-processor OS: A single processor system can execute only one process at any point of real time, though there might be many processes ready to be executed. A multiprocessing OS can support the execution of multiple processes at the same time. It uses multiple number of CPU. It is expensive in cost however; the processing speed will be faster. It is complex in its execution. Operating system like Unix, 64-bit edition of windows, server edition of windows, etc. are multiprocessing.

iii **Multiprogramming Operating System:** When two or more programs are in memory at the same time, sharing the processor is referred to the multiprogramming operating system. Multiprogramming assumes a single processor that is being shared. It increases CPU utilization by organizing jobs so that the CPU always has one to execute.

- The operating system keeps several jobs 111 memory at a time. This set of jobs is a subset of the jobs kept in the job pool. The operating system picks and begins to execute one of the job in the memory.

- Multiprogrammed systems provide an environment in which the various system resources are utilized effectively, but they do not provide for user interaction with the computer system.

- Jobs entering into the system are kept into the memory. Operating system picks the job and begins to execute one of the jobs in the memory. Having several programs in memory at the same time requires some form of memory management.

- Multiprogramming operating system monitors the state of all active programs and system resources. This ensures that the CPU is never idle unless there are no jobs.

Advantages

1. High CPU utilization.

2. It appears that many programs are allotted CPU almost simultaneously. **Disadvantages**

1. CPU scheduling is required.

To accommodate many jobs in memory, memory management is required.

iv Single-tasking versus Multitasking OS: A task is a function such as reading a file from disk, performing a math calculation, printing a document, or sending a request over the Internet to a Web server. Small and simple OSs can only manage a single task at a time. A single tasking OS allows only a single task to run at a time.

In a multitasking system more than one task can be performed at the same time but they are executed one after another through a single CPU by time sharing. For example: Windows, Linux, Mac, Unix, etc

Multi-threading: A program in execution is known as process. A process can be further divided into multiple sub-processes. These sub-processes are known as threads. A multi-threading OS can divide process into threads and execute those threads. This increases operating speed but also increases the complexity. For example: Unix, Server edition of Linux and windows.

Popular Operating Systems

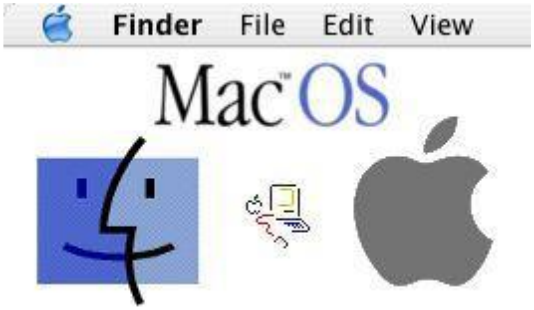
Initially computers had no operating systems. Every program needed full hardware specifications to run correctly as processor, memory and device management had to be done by the programs themselves. However, as sophisticated hardware and more complex application programs developed, operating systems became essential. As personal computers became popular among individuals and small businesses, demand for standard operating system grew. Let us look at some of the currently popular operating systems

 Microsoft Windows: Microsoft Windows is a series of software operating system based in graphical users' interfaces produced by Microsoft.

The different versions of Windows are: Windows 1.0, Windows 2.0, Windows 2000, Windows 95, Windows 98, Windows XP, Windows Vista, Windows 7

 Mobile Operating System: The mobile O.S is the Operating system that controls all mobile devices.
The different systems for mobiles are: Windows Mobile, Palms OS, BlackBerry OS, Symbian OS, and Android

 Mac OS: Mac OS is an operating system developed by Apple Computer Inc. Macintosh is popular because the graphical user interface, it was the integral and unnamed system software first introduced in 1984 but is usually referred to simply as the system software.
Mac OS can be divided into two families: The Mac OS Classic family and the Mac OS X operating system.



Network and system monitoring: Network consist of two or more systems communicating with each other. Basically, constitute four components:

- Protocol (communication rules)
- NIC (hardware for send and receive message)
- Cable (medium to connect)
- Hub/Switch (traffic control)
- Wireless network may not need cables or NIC

Network monitoring utility

- Helps to avoid bandwidth and server performance bottlenecks
- Helps to control and manage the network performance. Example, PRGT Network monitor ipMonitor

Security monitoring tool: Statistic has shown that 8 out 10 employees in the USA send and receive personal emails during working hours Similarly, 9 out of 10 employees visit websites that are not work related during working hours. Security monitoring tools are meant to log everything a user does, including website visited, keystrokes, internet connections etc. Example

- Spyagent
- Netvizor
- Network file monitor

Communication Software: A software that enables two or more computers to connect to each other via means like (modem, wireless, etc) for the purpose of information sharing operating systems like Windows/Linux/Dos have in-built communication software which must be configured with the ISP(Internet Service Providers).

Communication software includes:

- E-mail Software
- Wireless software
- Voicemail software
- Messaging software
- Paging software
- Internet Communication Software

Advantage of using Operating System

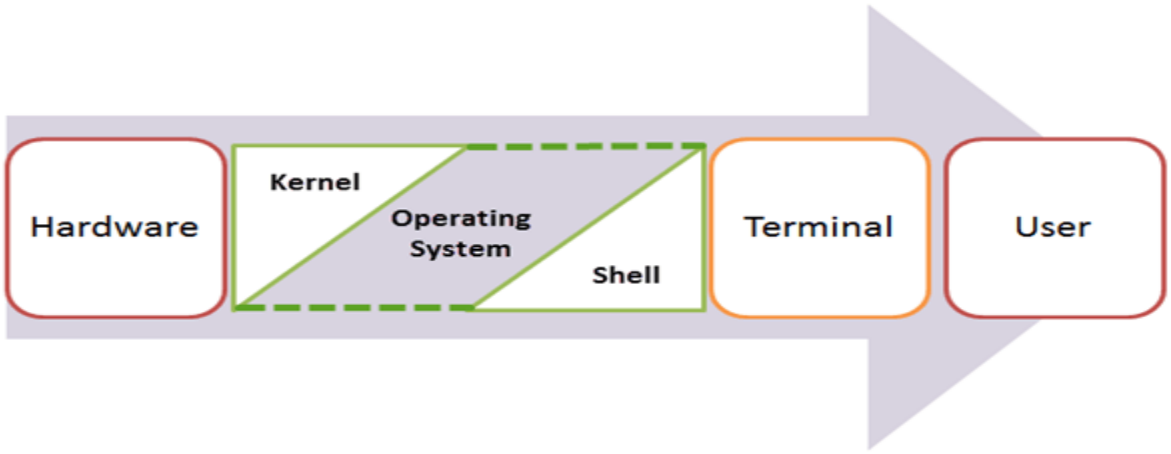
- Allows you to hide details of hardware by creating an abstraction
- Easy to use with a GUI
- Offers an environment in which a user may execute programs/applications
- The operating system must make sure that the computer system convenient to use
- Operating System acts as an intermediary among applications and the hardware components
- It provides the computer system resources with easy to use format
- Acts as an intermediator between all hardware's and software's of the system

Disadvantages of using Operating System

- If any issue occurs in OS, you may lose all the contents which have been stored in your system
- Operating system's software is quite expensive for small size organization which adds burden on them. Example Windows
- It is never entirely secure as a threat can occur at any time

What is a Kernel?

The kernel is the central component of a computer operating systems. The only job performed by the kernel is to the manage the communication between the software and the hardware. A Kernel is at the nucleus of a computer. It makes the communication between the hardware and software possible. While the Kernel is the innermost part of an operating system, a shell is the outermost one.



Features of Kennel

- Low-level scheduling of processes
- Inter-process communication
- Process synchronization
- Context switching

Types of Kernels

There are many types of kernels that exists, but among them, the two most popular kernels are:

1. **Monolithic**

A monolithic kernel is a single code or block of the program. It provides all the required services offered by the operating system. It is a simplistic design which creates a distinct communication layer between the hardware and software.

2. Microkernels

Microkernel manages all system resources. In this type of kernel, services are implemented in different address space. The user services are stored in user address space, and kernel services are stored under kernel address space. So, it helps to reduce the size of both the kernel and operating system

Deadlocks: Processes compete for physical and logical resources in the system (e.g. disks or files). Deadlocks affect the progress of processes by causing indefinite delays in resource allocation. Such delays have serious consequences for the response times of processes, idling and wastage of resources allocated to processes, and the performance of the system. Hence an OS must use resource allocation policies, which ensure an absence of deadlocks. This chapter characterizes the deadlock problem and describes the policies an OS can employ to ensure an absence of deadlocks.
DEFINITIONS
We define three events concerning resource allocation:
1. Resource request: A user process requests a resource prior to its use. This is done through an OS call. The OS analyses the request and determines whether the requested resource can be allocated to the process immediately. If not. The process remains blocked on the request till the resource is allocated.

2. Resource allocation: The OS allocates are source to a requesting process. The resource status information is updated and the state of the process is changed to ready. The process now becomes the holder of the resource.

3. Resource release: After completing resource usage, a user process releases the resource through an OS call. If another process is blocked on the resource, OS allocates the resource to it. If several processes are blocked on the resource, the OS uses some tie-breaking rule, e.g. FCFS allocation or allocation according to process priority, to perform the allocation.
Deadlock: A deadlock involving a set of processes D is a situation in which 1.Every process pi in D is blocked on some event ei Event ei can only be caused by some process (es) in D. If the event awaited by each process in D is the granting of some resource, it results in a resource deadlock. A communication deadlock occurs when the awaited events pertain to the receipt of inter-process messages, and synchronization deadlock when the awaited events concern the exchange of signals between processes. An OS is primarily concerned with resource deadlocks because allocation of resources is an OS responsibility. The other two forms of deadlock are seldom tackled by an OS.
HANDLING DEADLOCKS
Two fundamental approaches used for handling deadlocks are:

1 Detection and resolution of deadlocks
2. Avoidance of deadlocks.
In the former approach, the OS detects deadlock situations as and when they arise. It then performs some actions aimed at ensuring progress for some of the deadlocked processes. These actions constitute deadlock resolution. The latter approach focuses on avoiding the occurrence of deadlocks. This approach involves checking each resource request to ensure that it does not lead to a deadlock. The detection and resolution approach does not perform any such checks. The choice of the deadlock handling approach would depend on the relative costs of the approach, and its consequences for user processes.

DEADLOCK DETECTION AND RESOLUTION

The deadlock characterization developed in the previous section is not very useful in practice for two reasons. First, it involves the overheads of building and maintaining an RRAG. Second, it restricts each resource request to a single resource unit of one or more resource classes. Due to these limitations, deadlock detection cannot be implemented merely as the determination of a graph property. For a practical implementation, the definition can be interpreted as follows: A set of blocked processes D is deadlocked if does not exist any sequence of resource allocations and resource releases in the system whereby each process in D can complete. The OS must determine this fact through exhaustive analysis.

Deadlock analysis is performed by simulating the completion of a running process. In the simulation it is assumed that a running process completes without making additional resource requests. On completion, the process releases all resources allocated to it. These resources are allocated to a blocked process only if the process can enter the running state. The simulation terminates in one of two situations—either all blocked processes become running and complete, or some set B of blocked processes cannot be allocated their requested resources. In the former case no deadlock exists in the system at the time when deadlock analysis is performed, while in the latter case processes in B are deadlocked. Deadlock Resolution

Given a set of deadlocked processes D, deadlock resolution implies breaking the deadlock to ensure progress for some processes {pi} £ D. This can be achieved by satisfyin**g** the resource request of a process pi in one of two ways:

1. Terminate some processes {pj} e D to free the resources required by pi. (We call each pj a victim of deadlock resolution.) 2. Add a new unit of the resource requested by pi. Note that deadlock resolution only ensures some progress for pi. It does not guarantee that a pi would run to completion. That would depend on the behaviour of processes after resolution.

CP/M

Control Program/Microcomputer. An operating system created by Gary Kildall, the founder of Digital Research. Created for the old 8-bit microcomputers that used the 8080, 8085, and Z-80 microprocessors. They may be dynamically reassigned among a collection of active programs in different stages of execution.

Several variations of both serial and multi programmed operating systems exist.

Spooling:
Acronym for simultaneous peripheral operations on line. Spooling refers to putting jobs in a buffer, a special area in memory or on a disk where a device can access them when it is ready.

Spooling is useful because device access data at different rates. The buffer provides a waiting station where data can rest while the slower device catches up.

• Computer can perform I/O in parallel with computation, it becomes possible to have the computer read a deck of cards to a tape, drum or disk and to write out to a tape printer while it was computing. This process is called **spooling.**

• The most common spooling application is print spooling. In print spooling, documents are loaded into a buffer and then the printer pulls them off the buffer at its own rate.

• Spooling is also used for processing data at remote sites. The CPU sends the data via communications path to a remote printer. Spooling overlaps the I/O of one job with the computation of other jobs.

• One difficulty with simple batch systems is that the computer still needs to read the deck of cards before it can begin to execute the job. This means that the CPU is idle during these relatively slow operations. • Spooling batch systems were the first and are the simplest of the multiprogramming systems.

Advantages of Spooling:
1. The spooling operation uses a disk as a very large buffer.

2. Spooling is however capable of overlapping I/O operation for one job with processor operations for another job.

Computing Environments:
• Different types of computing environments are:

a. Traditional computing
b. Web based computing
c. Embedded computing

Typical office environment uses traditional computing. Normal PC is used in traditional computing.

• Web technology also uses traditional computing environment. Network computers are essentially terminals that understand web based computing. In domestic application, most of user had a single computer with Internet connection. Cost of the accessing Internet is high.

• Web based computing has increased the emphasis on networking. Web based computing uses PC, handheld PDA and cell phones. One of the features of this type is load balancing. In load balancing, network connection is distributed among a pool of similar servers.

• Embedded computing uses real time operating systems. Application of embedded computing is car engines, manufacturing robots to VCR and microwave ovens. This type of system provides limited features.

Architecture and Organization

• **Architecture** is the design of the system visible to the assembly level programmer.
What instructions

How many registers

Memory addressing scheme

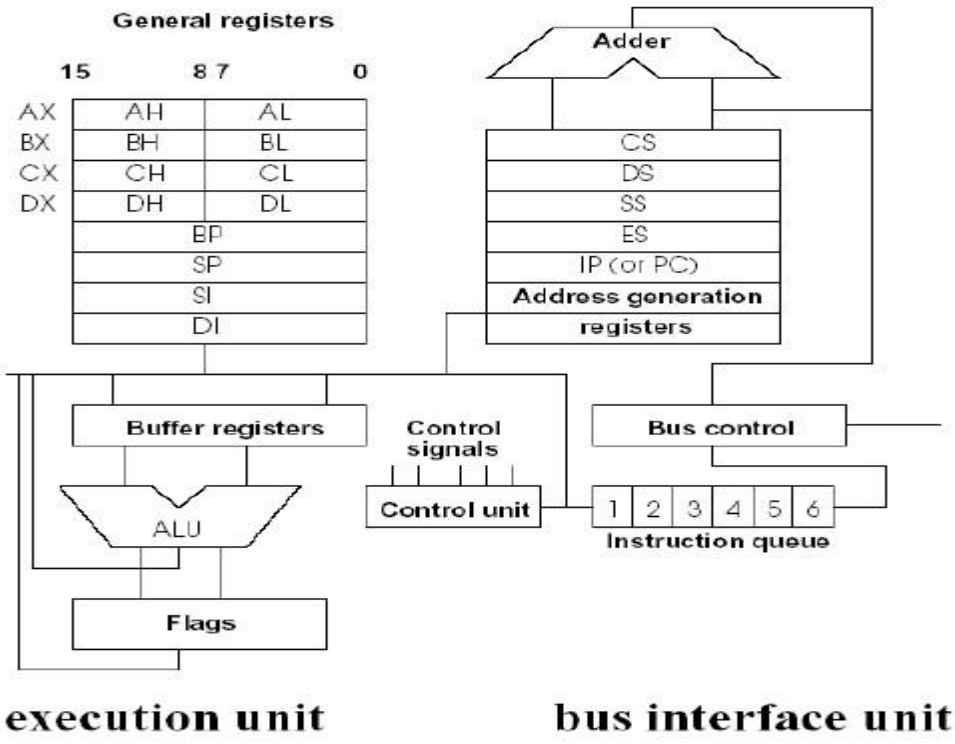
• **Organization** is how the architecture is implemented.
How much cache memory

Microcode or direct hardware

Implementation technolog**y**

(8086 Architecture)

Hardware Organization



On the structural scheme of the i8086 processor we can see two separate asynchronous processing units. The execution unit (EU) executes instructions; the bus interface unit (BIU) fetches instructions, reads operands, and writes results. The two units can operate almost independently of one another and are able, under most circumstances, to extensively overlap instruction fetch with execution. The result is that, in most cases, the time normally required to fetch instructions "disappears" because the EU executes instructions that have already been fetched by BIU. Of course nothing special, but remember the time when i8086 was designed.

Execution Unit

The execution unit consists of general registers, buffer registers, control unit, arithmetic/logic unit, and flag register. The ALU maintains the CPU status and control flags and manipulates the general registers and instruction operands. The EU is not connected to the system bus. It obtains instructions from a queue maintained by the BIU. Likewise, when an instruction requires access to memory or to a peripheral device, the EU requests the BIU to obtain or store the data. The EU manipulates only with 16-bit addresses (effective addresses). An address relocation that enables the EU access to the full megabyte is performed by BIU.

Bus Interface Unit

The bus interface unit performs all bus operations for the EU. Data is transferred between the CPU and memory or I/O devices upon demand from the EU. During periods when the EU is busy executing instructions, the BIU fetches more instructions from memory. The instructions are stored in an internal RAM array called the instruction stream queue. The 8086 queue can store up to six instruction bytes. This allows the BIU to keep the EU supplied with prefetched instructions under most conditions. The BIU of 8086 does not initiate a fetch until there are two empty bytes in its queue. The BIU normally obtains two instruction bytes per fetch, but if a program transfer forces fetching from an odd address, the 8086 BIU automatically reads one byte from the odd address and then resumes fetching two-byte words from the subsequent even addresses. Under most circumstances the queue contains at least one byte of the instruction stream and the EU does not have to wait for instructions to be fetched. The instructions in the queue are the next logical instructions so long as execution proceeds serially. If the EU executes an instruction that transfers control to another location, the BIU resets the queue, fetches the instruction from the new address, passes it immediately to the EU, and then begins refilling the queue from the new caption. In addition, the BIU suspends instruction fetching whenever the EU requests a memory or I/O read or write (except that a fetch already in progress is completed before executing the EU's bus request).

The Details of the Architecture

Registers

The general registers of the 8086 are divided into two sets of four 16-bit registers each. The data registers and the pointer and index registers. The data registers' upper and lower halves are separately addressable. In other words, each data register can be used interchangeably as a 16-bit register or as two 8-bit registers. The data registers can be used without constraint in most arithmetic and logic operations. Some instructions use certain registers implicitly thus allowing compact yet powerful encoding. The pointer and index registers can be used only as 16-bit

registers. They can also participate in most arithmetic and logic operations. In fact all eight general registers fit the definition of "accumulator" as used in first and second generation microprocessors. The pointer and index registers (except BP) are also used implicitly in some instructions. The segment registers contain the base addresses of logical segments in the 8086 memory space. The CPU has direct access to four segments at a time. The CS register points to the current code segment; instructions are fetched from this segment. The SS points to the current stack segment; stack operations are performed on locations in this segment. The DS register points to the current data segment; it generally contains program variables. The ES points to the current extra segment; which also is typically used for data storage. The IP (instruction pointer) is updated by the BIU so that it contains the offset of the next instruction from the beginning of the current code segment. During normal execution IP contains the offset of the next instruction to be *fetched* by the BIU; whenever IP is saved on the stack, however, it is first automatically adjusted to point to the next instruction to be *executed.* Programs do not have direct access to the IP.
There are eight 16-bit general registers.

The data registers:
AX (AH and AL)
BX (BH and BL)
CX (CH and CL)
DX (DH and DL)

The pointer and index registers: BP, SP, SI, DI
The upper and lower halves of the data registers are separately addressable. Memory space is divided into logical segments up to 64k bytes each. The CPU has direct access to four segments at a time; their base addresses are contained in the segment registers **CS, DS, SS, ES. CS** = code segment;

DS = data segment; **SS** = stack segment;
ES = extra segment;

Flags are maintained in the flag register depending on the result of the arithmetic or logic operation. A group of instructions is available that allows a program to alter its execution depending on the state of the flags, that is, on the result of a prior operation. There are:
• **AF** (the auxiliary carry flag) used by decimal arithmetic instructions. Indicates carry out from the low nibble of the 8-bit quantity to the high nibble, or borrow from the high nibble into the low nibble.

• **CF** (the carry flag) indicates that there has been carry out of , or a borrow into, the high-order bit of the result.

• **OF** (the overflow flag) indicates that an arithmetic overflow has occurred.

• **SF** (the sign flag) indicates the sign of the result (high-order bit is set, the result is negative).

• **PF** (the parity flag) indicates that the result has an even parity, an even number of 1-bits.

• **ZF** (the zero flag) indicates that the result of the operation is 0. Three additional control flags can be set and cleared by programs to alter processor operations:

• **DF** (the direction flag) causes string instructions to auto-decrement if it is set and to auto-increment if it is cleared.

• **IF** (the interrupt enable flag) allows the CPU to recognize external interrupts.

• **TF** (the trap flag) puts the processor into single-step mode for debugging.

Memory Organization

The 8086 can accommodate up to 1,048,576 bytes of memory. From the storage point of view, the memory space is organized as array of 8-bit bytes. Instructions, byte data and word data may be freely stored at any byte address without regard for alignment. The Intel convention is that the most-significant byte of word data is stored in the higher memory location. A special class of data (pointers) is stored as double words. The lower addressed word of a pointer contains an offset value, and the higher-addressed word contains a segment base address. Each word is stored following the above convention. The i8086 programs "view" the megabyte of memory space as a group of segments that are defined by the application. A segment is a logical unit of memory that may be up to 64k bytes long. Each segment is made up of contiguous memory locations and is an independent separately addressable unit. The software must assign to every segment a base address, which is its starting location in the memory space. All segments begin on 16-byte memory boundaries called paragraphs. The segment registers point to the four currently addressable segments. To obtain code and data from other segments, program must change the content of segment registers to point to the desired segments. Every memory location has its physical address and its logical address. A physical address is the 20-bit value that uniquely identifies each byte location in the memory space. Physical addresses may range from 0H to FFFFFH. All exchanges between the CPU and memory components use physical addresses. However, programs deal with logical rather than physical addresses. A logical address consists of a segment base and offset value. The logical to physical address translation is done by BIU whenever it accesses memory. The BIU shifts segment base by 4 to the left and adds the offset to this value. Thus we obtain 20-bit physical address and get the explanation for 16-byte memory boundaries for the segment base beginning. The offset of the memory variable is calculated by the EU depending on the addressing modes and is called the operand's effective address (EA). Stack is implemented in memory and is located by the stack segment register and the stack pointer register. An item is pushed onto the stack by decrementing SP by 2 and writing the item at the new top of stack (TOS). An item is popped off the stack by copying it from TOS and then incrementing SP by 2. The memory locations 0H through 7FH are dedicated for interrupt vector table, and locations FFFF0H through FFFFFH are dedicated for system reset.

Input/Output

The 8086 I/O space can accommodate up to 64k 8-bit ports or up to 32k 16-bit ports. The IN and OUT instructions transfer data between the accumulator and ports located in I/O space. The I/O space is not segmented; to access a port, the BIU simply places the port address on the lower 16

lines of the address bus. I/O devices may also be placed in the 8086 memory space. As long as the devices respond like the memory components, the CPU does not know the difference. This adds programming flexibility, and is paid by longer execution of memory oriented instructions.

Processor Control and Monitoring

The interrupt system of the 8086 is based on the interrupt vector table which is located from 0H through 7FH (dedicated) and from 80H through 3FFH (user available). Every interrupt is assigned a type code that identifies it to the CPU. By multiplying (type * 4), the CPU calculates the location of the correct entry for a given interrupt. Every table entry is 4 bytes long and contains the offset and the segment base (pointer) of the corresponding interrupt procedure that should be executed. After system reset all segments are initialized to 0H except CS which is initialized to FFFFH. Since, the processor executes the first instruction from absolute memory location FFFF0H. This location normally contains an intersegment direct JMP instruction whose target is the actual beginning of the system program.

EXCEPTION HANDLING

This part examines Java's exception-handling mechanism. An *exception* is an abnormal condition that arises in a code sequence at run time. In other words, an exception is a run-time error. In computer languages that do not support exception handling, errors must be checked and handled manually—typically through the use of error codes, and so on. This approach is as cumbersome as it is troublesome. Java's exception handling avoids these problems and, in the process, brings run-time error management into the object oriented world.

An exception is an event that occurs during the execution of a program that disrupts the normal flow of instructions.

Exception-Handling Fundamentals

•***Exception****:* an abnormal condition that arises in a code sequence at run time/ run time error

–Java exception is an object that describes an exceptional (that is, error) condition

–When an exceptional condition arises, an object representing that exception is created and ***thrown*** in the method that caused the error

•Exception can be –generated by the Java run-time system (relate to fundamental errors that violate the rules of the Java language)

–Manually generated (typically used to report some error condition to the caller of a method)

Keywords for exception handling

•**try** block contains program statements that are to be to monitored for exceptions

– If an exception occurs within the **try** block, it is thrown

•**catch** block contain statements that catch this exception and handle it in some rational manner

–System-generated exceptions are automatically thrown by the Java runtime system.

•**throw** is used to manually throw an exception

•**throws** clause is used to specify any exception that is thrown out of a method

•**finally** block contains code that absolutely must be executed after a **try** block completes

General form of an exception-handling block

```
try {
// block of code to monitor for errors
}
catch (ExceptionType1 exOb) {
// exception handler for ExceptionType1
}
catch (ExceptionType2 exOb) {
// exception handler for ExceptionType2
}
// ...
finally {
// block of code to be executed after try block ends
}
```

Advantages

• Code that handles errors and unusual events can be separated from the normal code.

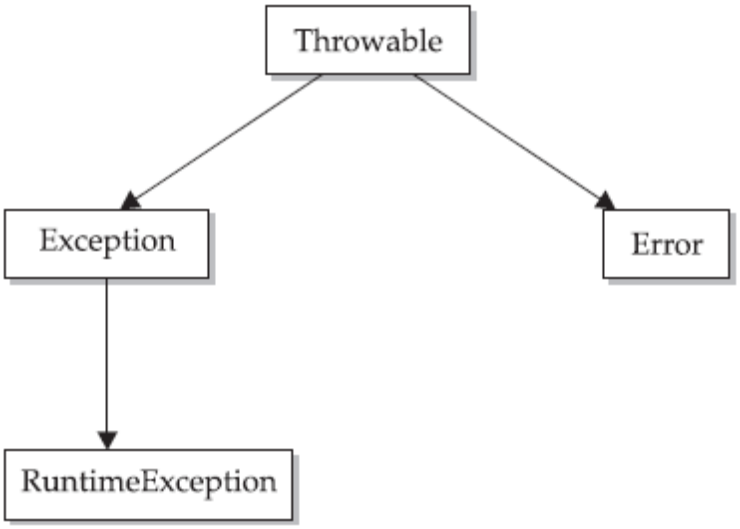
• Errors automatically propagate up the calling chain until they are handled.

• Errors and special conditions can be classified and grouped according to common properties.

Exception Types

All exception types are subclasses of the built-in class **Throwable**. Thus, **Throwable** is at the top of the exception class hierarchy. Immediately below **Throwable** are two subclasses that partition exceptions into two distinct branches. One branch is headed by **Exception**. This class is used for exceptional conditions that user programs should catch. This is also the class that you will subclass to create your own custom exception types. There is an important subclass of **Exception**, called **RuntimeException**. Exceptions of this type are automatically defined for the programs that you write and include things such as division by zero and invalid array indexing.

The other branch is topped by **Error**, which defines exceptions that are not expected to be caught under normal circumstances by your program. Exceptions of type **Error** are used by the Java run-time system to indicate errors having to do with the run-time environment, itself. Stack overflow is an example of such an error. Here, we'll not be dealing with exceptions of type **Error**, because these are typically created in response to catastrophic failures that cannot usually be handled by your program.



Using try and catch

•Exception handling benefits

–allows you to fix the error

–prevents the program from automatically terminating

Before you learn how to handle exceptions in your program, it is useful to see what happens when you don't handle them. This small program includes an expression that intentionally causes a divide-by-zero error:

```
class Exc1 {
public static void main(String args[]) {
int d, a;
try { // monitor a block of code.
d = 0;
a = 42 / d;
System.out.println("This will not be printed.");
} catch (ArithmeticException e) {
// catch divide-by-zero error
System.out.println("Division by zero.");
}
System.out.println("After catch statement.");
}
}
```

When the Java run-time system detects the attempt to divide by zero, it constructs a new exception object and then *throws* this exception. This causes the execution of **Exc0** to stop, because once an exception has been thrown, it must be *caught* by an exception handler and dealt with immediately. In this example, we haven't supplied any exception handlers of our own, so the exception is caught by the default handler provided by the Java run-time system. Any exception that is not caught by your program will ultimately be processed by the default handler. The default handler displays a

string describing the exception, prints a stack trace from the point at which the exception occurred, and terminates the program.

Using try and catch

Although the default exception handler provided by the Java run-time system is useful for debugging, you will usually want to handle an exception yourself. Doing so provides two benefits. First, it allows you to fix the error. Second, it prevents the program from automatically terminating. Most users would be confused (to say the least) if your program stopped running and printed a stack trace whenever an error occurred! Fortunately, it is quite easy to prevent this.

To guard against and handle a run-time error, simply enclose the code that you want to monitor inside a **try** block. Immediately following the **try** block, include a **catch** clause that specifies the exception type that you wish to catch. To illustrate how easily this can be done, the following program includes a **try** block and a **catch** clause that processes the **ArithmeticException** generated by the division-by-zero error:

```
class Exc2 {
public static void main(String args[]) {
int d, a;
try { // monitor a block of code.
d = 0;
a = 42 / d;
System.out.println("This will not be printed.");
} catch (ArithmeticException e) { // catch divide-by-zero error
System.out.println("Division by zero.");
}
System.out.println("After catch statement.");
}
}
```

This program generates the following output:

```
Division by zero.
After catch statement.
```

Notice that the call to **println( )** inside the **try** block is never executed. Once an exception is thrown, program control transfers out of the **try** block into the **catch** block. Put differently, **catch** is not "called," so execution never "returns" to the **try** block from a **catch**. Thus, the line "This will not be printed." is not displayed. Once the **catch** statement has executed, program control continues with the next line in the program following the entire **try**/**catch** mechanism.

A **try** and its **catch** statement form a unit. The scope of the **catch** clause is restricted to those statements specified by the immediately preceding **try** statement. A **catch** statement cannot catch an exception thrown by another **try** statement (except in the case of nested **try** statements). The statements that are protected by **try** must be surrounded by curly braces. (That is, they must be within a block.) You cannot use **try** on a single statement.

NOTE:
Java defines several other types of exceptions that relate to its various class libraries.

Using Exceptions

Exception handling provides a powerful mechanism for controlling complex programs that have many dynamic run-time characteristics. It is important to think of **try** and **catch** as clean (simple) way to handle errors and unusual boundary conditions in your program's logic. Unlike some other languages in which error return codes are used to indicate failure, Java uses exceptions. Thus, when a method can fail, have it throw an exception. This is a cleaner way to handle failure modes. One last point: Java's exception-handling statements should not be considered a general mechanism for nonlocal branching. If you do so, it will only confuse your code and make it hard to maintain.