CHAPTER 1

Introduction to Programming, Algorithms & Flow Charts

PROGRAM & PROGRAMMING

- A program is a set of logically related instructions that is arranged in a sequence that directs the computer in solving a problem.
- The process of writing a program is called programming.

- Software is a collection of computer programs and related data that provides the instructions for telling a computer what to do and how to do it.
- Computer software can be broadly classified into two categories :

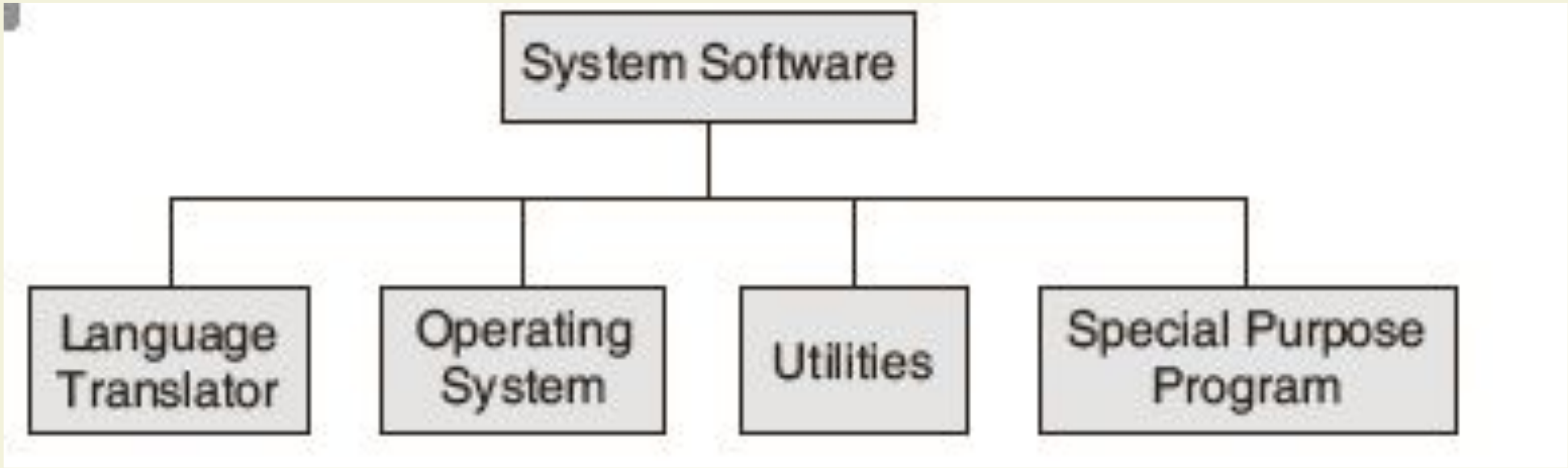
  (a) . system software

  &

  (b) . application software

SYSTEM SOFTWARE

- System software is a collection of programs that interfaces with the hardware.
- Categories of system software :


System Software

Language Translator

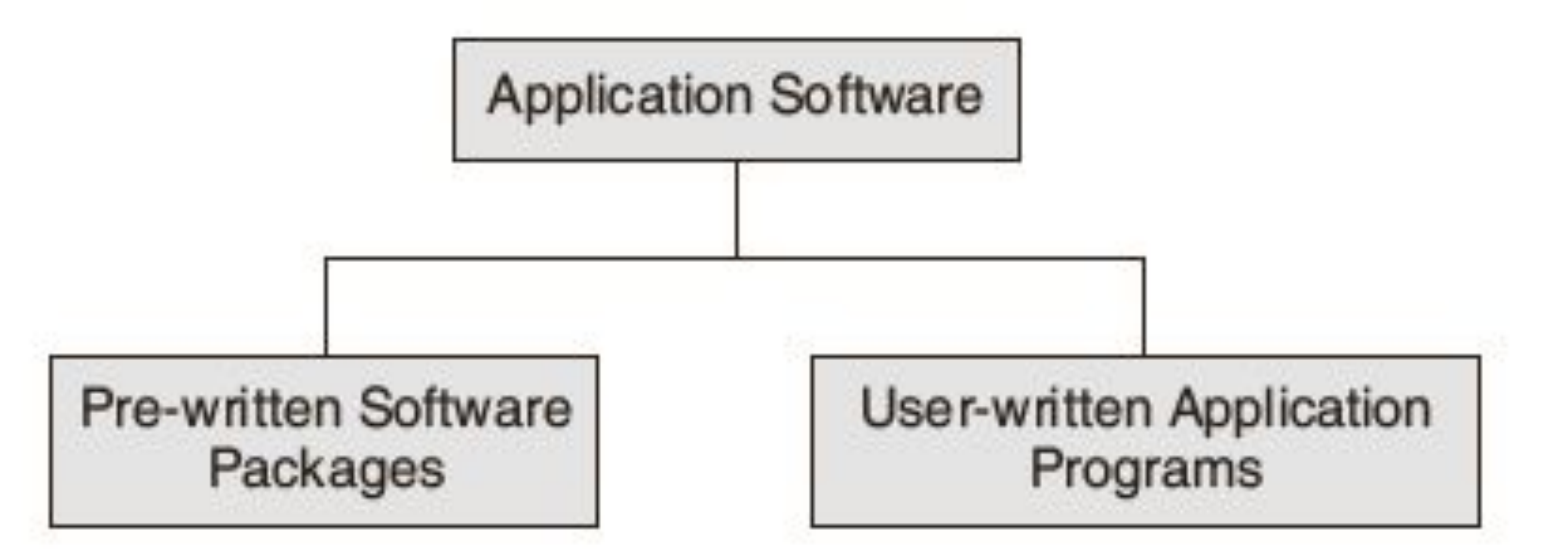
Operating System

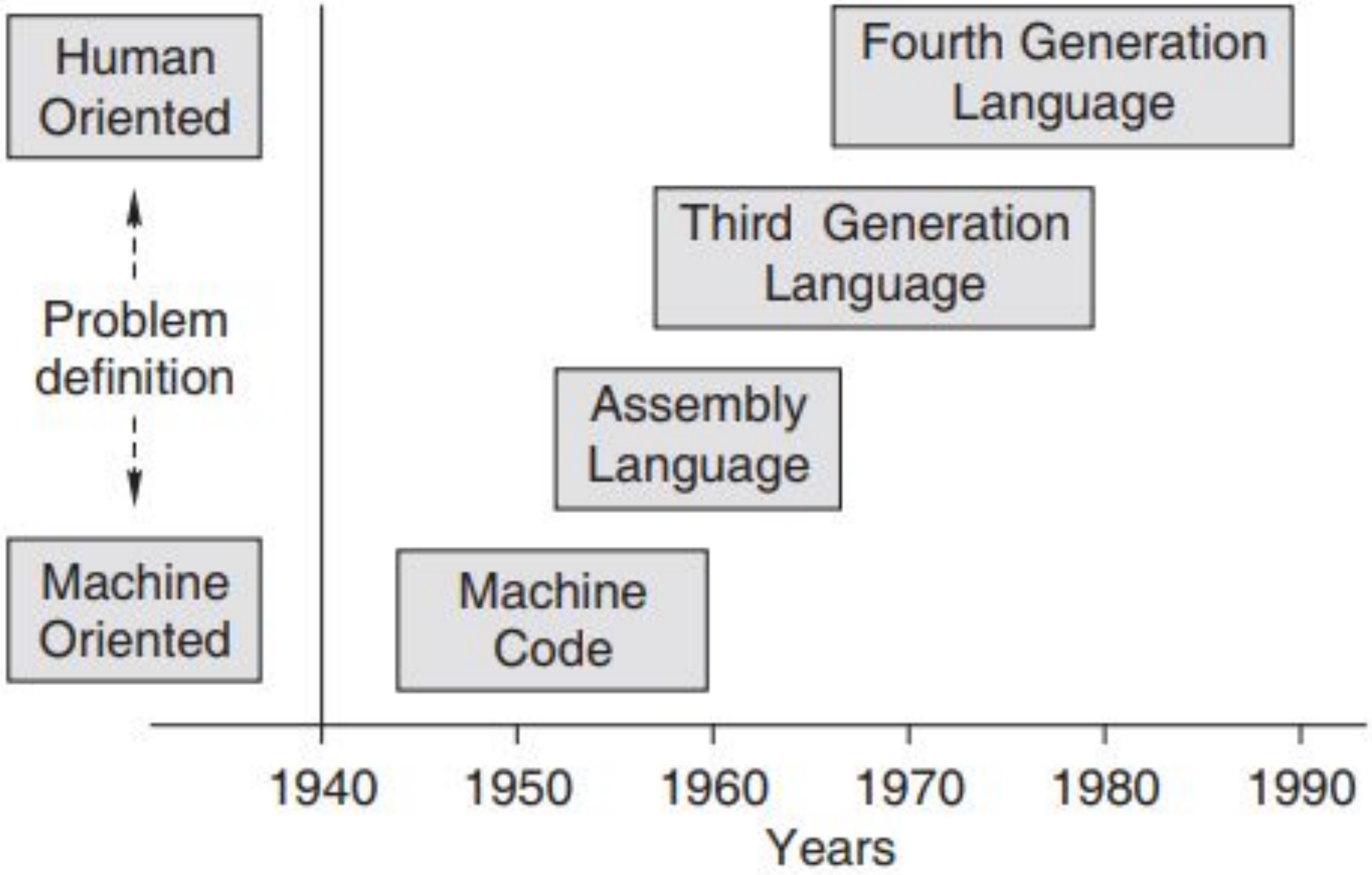
Utilities

Special Purpose Program

APPLICATION SOFTWARE

- Application software is written to enable the computer to solve a specific data processing task.
- Categories of application software :


Application Software
Pre-written Software Packages
User-written Application Programs


Human Oriented
Problem definition
Machine Oriented
Fourth Generation Language
Third Generation Language
Assembly Language
Machine Code
1940
1950
1960
1970
1980
1990
Years

PROGRAMMING LANGUAGE

- A programming language is composed of a set of instructions in a language understandable to the programmer and recognizable by a computer.

- Programming languages can be classified as

  (a) High-level language - BASIC, COBOL & FORTRAN(application programs).

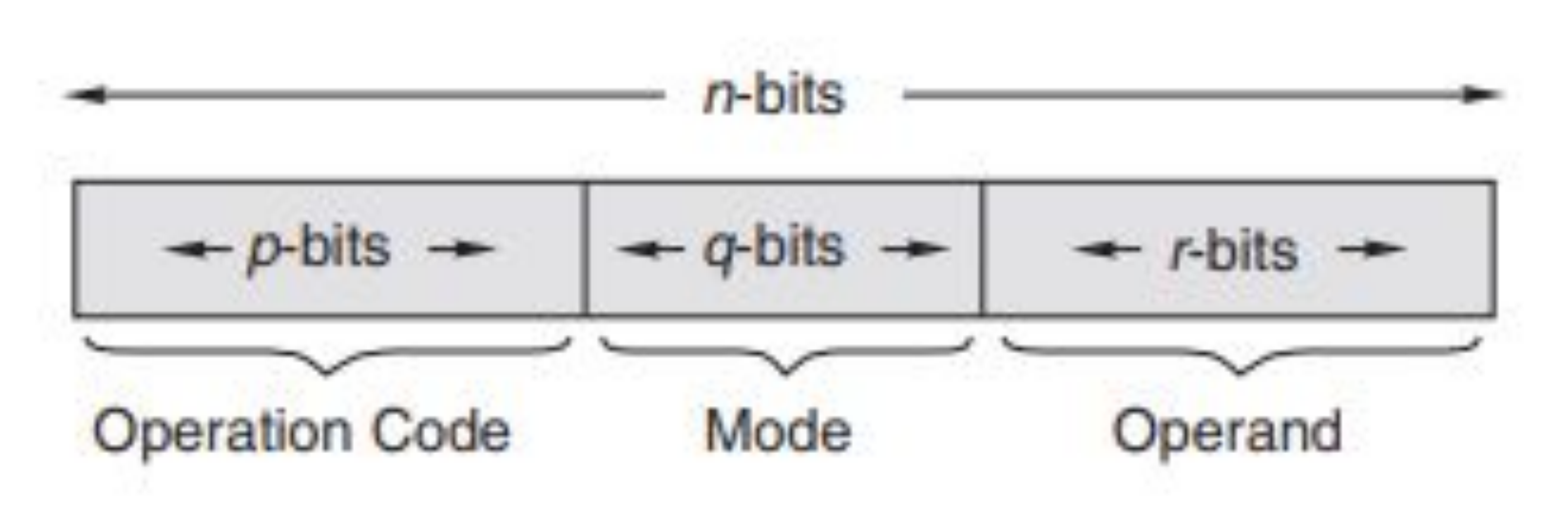
  (b) Middle level language - C (application & system programs).

  (c) Low level language – assembly language (system programs).

Machine Language

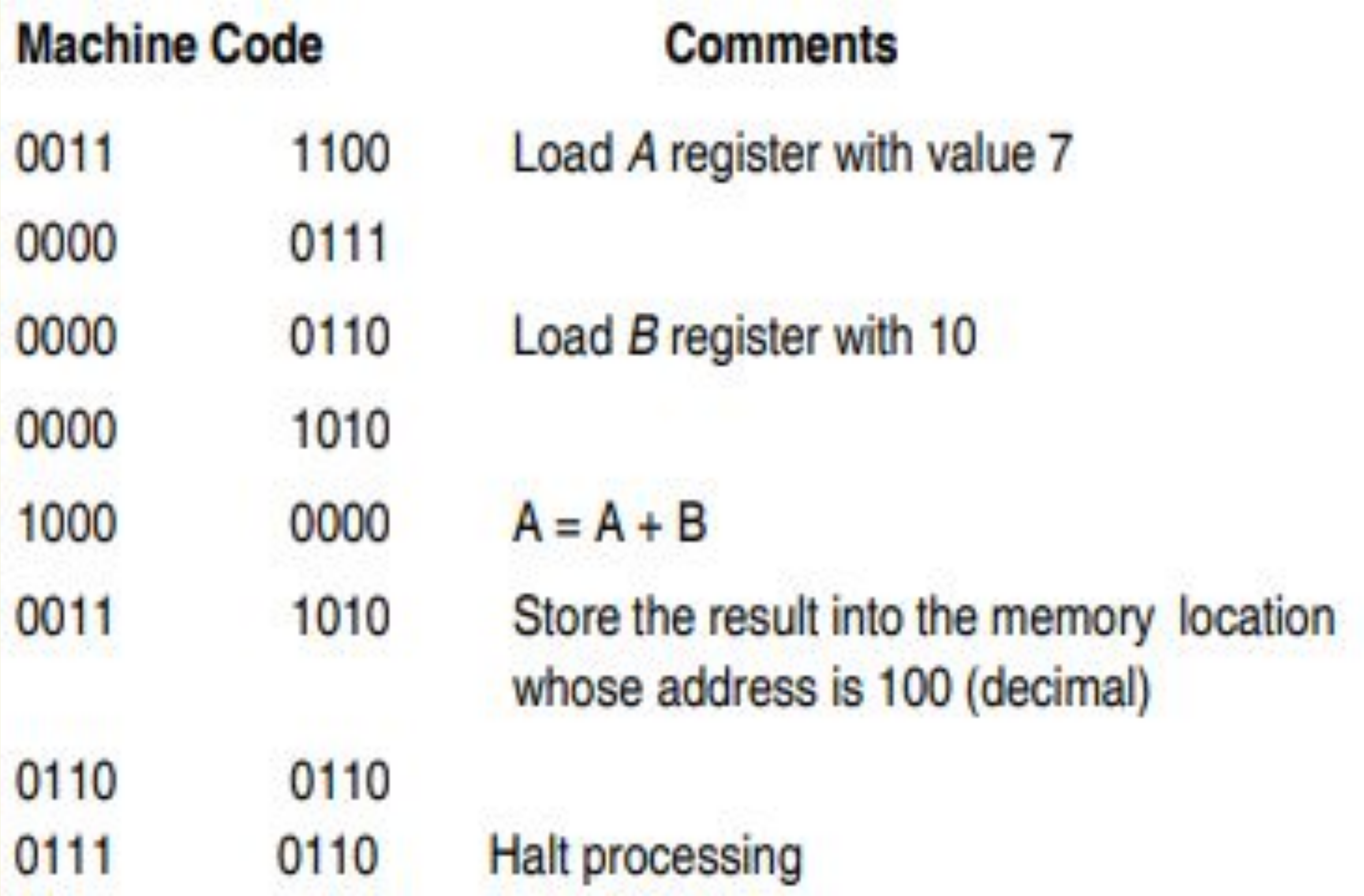
- This is a sequence of instructions written in the form of binary numbers consisting of 1's and 0's to which the computer responds directly.
- The machine language is also referred to as the machine code, although the term is used more broadly to refer to any program text.
- A machine language instruction generally has three parts
  - The first part is the command or operation code that conveys to the computer what function has to be performed by the instruction. All computers have operation codes for functions such as adding, subtracting and moving.
  - The second part of the instruction either specifies that the operand contains data on which the operation has to be performed or it specifies that the operand contains a location, the contents of which have to be subjected to the operation.
- Machine language is considered to be the first generation language (1GL)

Machine Language



- Difficult to use
- Machine dependent
- Error prone
- Difficult to debug and modify

Machine Language



Machine Code		Comments
0011	1100	Load *A* register with value 7
0000	0111	
0000	0110	Load *B* register with 10
0000	1010	
1000	0000	A = A + B
0011	1010	Store the result into the memory location whose address is 100 (decimal)
0110	0110	
0111	0110	Halt processing

- An example of a machine language program for adding two numbers.

Advantage of machine language

- The CPU directly understands machine instructions, and hence no translation is required.
- Therefore, the computer directly starts executing the machine language instructions, and it takes less execution time.

Disadvantages of machine language

- Difficult to use
  - It is difficult to understand and develop a program using machine language. For anybody checking such a program, it would be difficult to forecast the output when it is executed.
  - Nevertheless, computer hardware recognizes only this type of instruction code.
- Error prone
  - It is hard to understand and remember the various combinations of 1's and 0's representing data and instructions.
  - This makes it difficult for a programmer to concentrate fully on the logic of the problem, thus frequently causing errors.
- Difficult to debug and modify
  - Checking machine instructions to locate errors are about as tedious as writing the instructions.
  - Further, modifying such a program is highly problematic.

Disadvantages of machine language

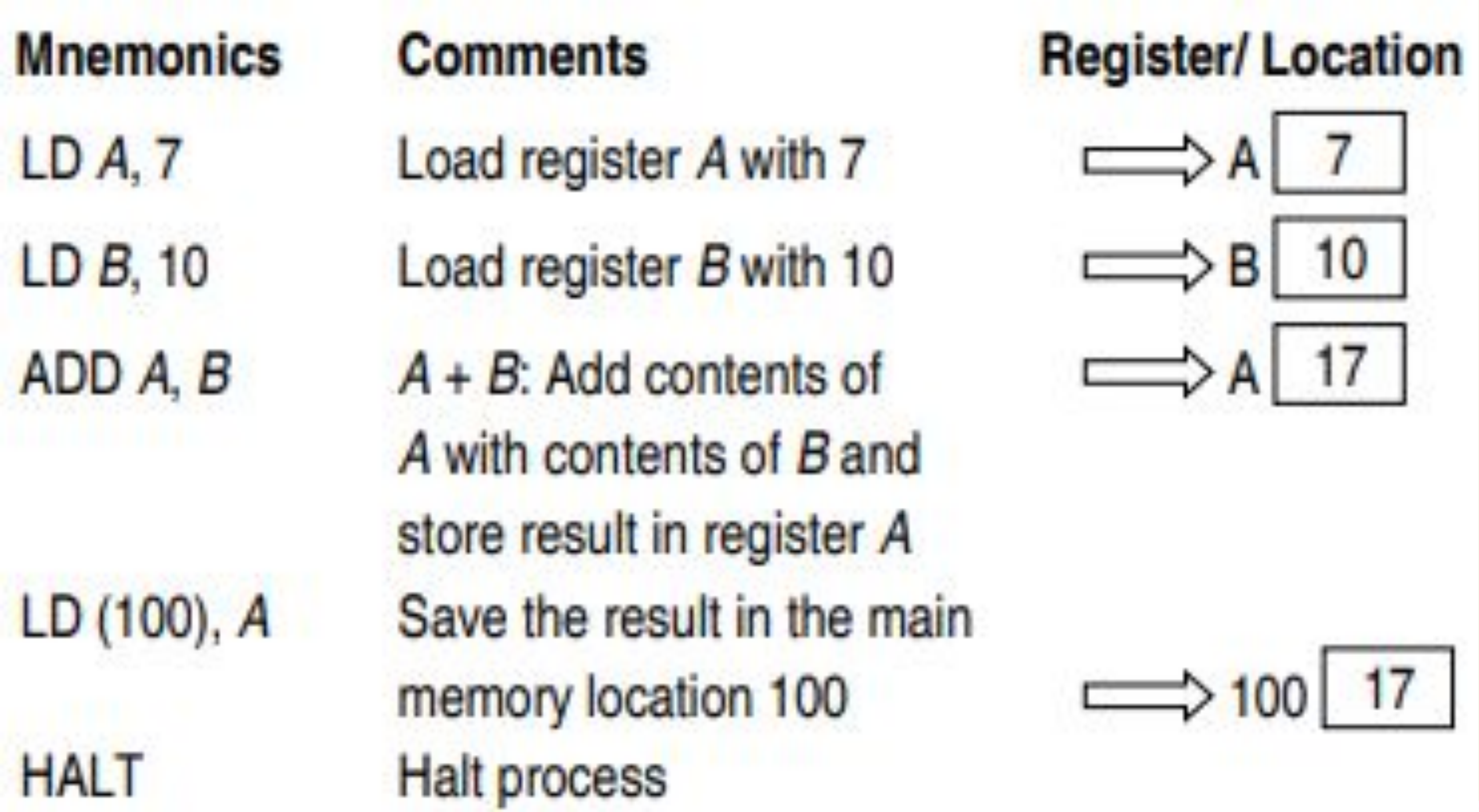
- Machine dependent
  - The programmer has to remember machine characteristics while preparing a program.
  - As the internal design of the computer is different across types, which in turn is determined by the actual design or construction of the ALU, CU, and size of the word length of the memory unit, the machine language also varies from one type of computer to another.
  - Hence, it is important to note that after becoming proficient in the machine code of a particular computer, the programmer may be required to learn a new machine code and would have to write all the existing programs again in case the computer system is changed

Assembly Language

- When symbols such as letters, digits, or special characters are employed for the operation, operand, and other parts of the instruction code, the representation is called an assembly language instruction.
- Such representations are known as mnemonic codes; they are used instead of binary codes.
- A program written with mnemonic codes forms an assembly language program. This is considered to be a second generation language (2GL).
- Machine and assembly languages are referred to as low-level languages since the coding for a problem is at the individual instruction level.
- Each computer has its own assembly language that is dependent upon the internal architecture of the processor.

Assembly Language

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



Mnemonics	Comments	Register/ Location
LD *A*, 7	Load register *A* with 7	⇨ A 7
LD *B*, 10	Load register *B* with 10	⇨ B 10
ADD *A*, *B*	*A* + *B*: Add contents of *A* with contents of *B* and store result in register *A*	⇨ A 17
LD (100), *A*	Save the result in the main memory location 100	⇨ 100 17
HALT	Halt process	

Assembly Language

- An assembler is a translator that takes input in the form of the assembly language program and produces machine language code as its output.
- An instruction word consists of parts,
  - the Opcode (Operation Code) part indicates the operation to be performed by the instruction and
  - the mode and operand parts convey the address of the data to be found or stored.

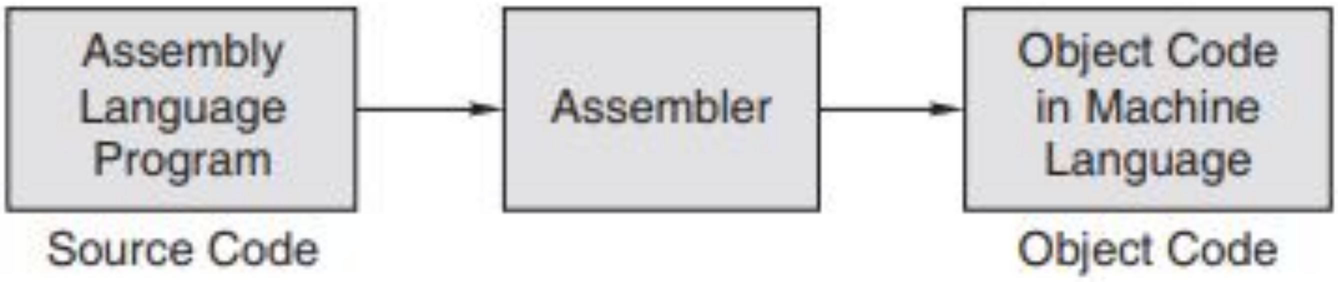


Figure 1.6 Assembler

Advantage of assembly language

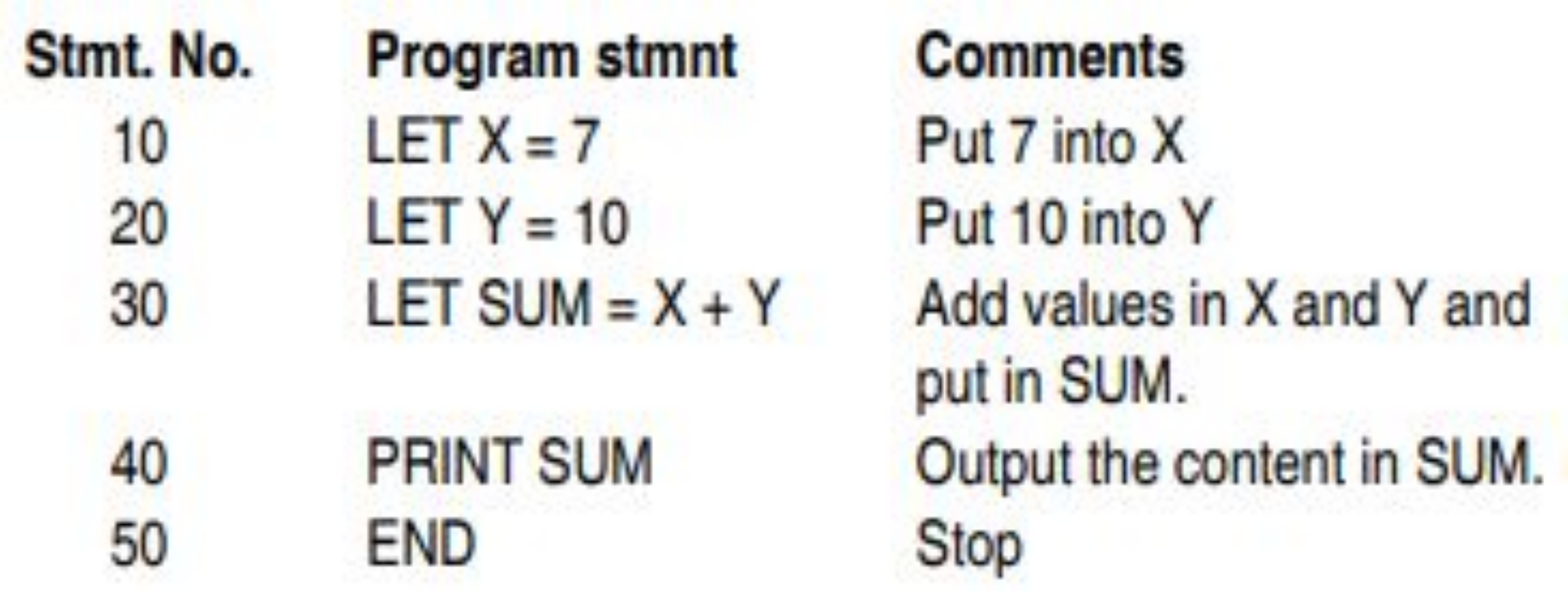
- Writing a program in assembly language is more convenient than writing one in machine language.
- Instead of binary sequence, as in machine language, a program in assembly language is written in the form of symbolic instructions.
- This gives the assembly language program improved readability

Disadvantages of assembly language

- Assembly language is specific to a particular machine architecture, i.e., machine dependent. Assembly languages are designed for a specific make and model of a microprocessor.
  - This means that assembly language programs written for one processor will not work on a different processor if it is architecturally different.
  - That is why an assembly language program is not portable.
- Programming is difficult and time consuming.
- The programmer should know all about the logical structure of the computer.

High Level Language

A program written in BASIC, to obtain the sum of two numbers, is shown below



Stmt. No.	Program stmnt	Comments
10	LET X = 7	Put 7 into X
20	LET Y = 10	Put 10 into Y
30	LET SUM = X + Y	Add values in X and Y and put in SUM.
40	PRINT SUM	Output the content in SUM.
50	END	Stop

Advantages of high-level programming languages

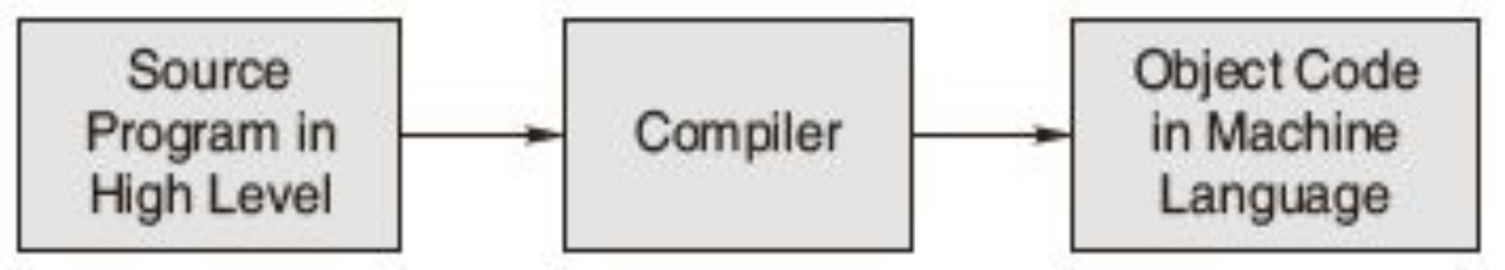
- Readability
  - Programs written in these languages are more readable than those written in assembly and machine languages.
- Portability
  - High-level programming languages can be run on different machines with little or no change. It is, therefore, possible to exchange software, leading to creation of program libraries.
- Easy debugging
  - Errors can be easily detected and removed.Ease in the development of software Since the commands of these programming languages are closer to the English language, software can be developed with ease.
- High-level languages are also called third generation languages (3GLs).

COMPILER

- For executing a program written in a high-level language, it must be first translated into a form the machine can understand. This is done by a software called the *compiler.*
- The compiling process consists of two steps:

  a . The analysis of the source program and

  b . The synthesis of the object program in the machine language of the specified machine.
- Compiler action :



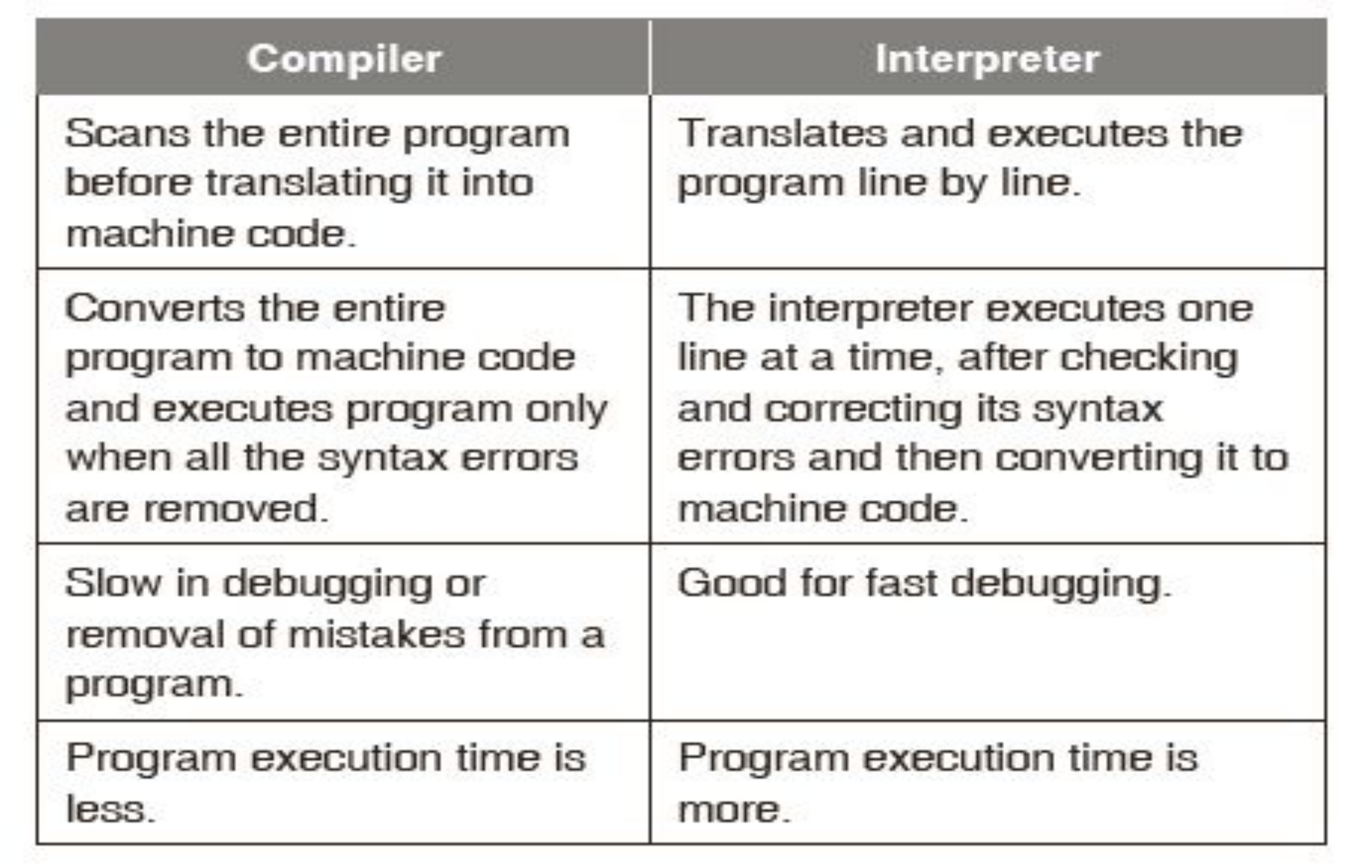
Source Program in High Level → Compiler → Object Code in Machine Language

COMPILER

- During the process of translation, the compiler reads the source program statement- wise and checks for syntax errors.
- In case of any error, the computer generates a printout of the same. This action is known as diagnostics.
- There is another type of software that also does translation. This is called an interpreter.
- The compiler and interpreter have different approaches to translation.

INTERPRETER

- During the process of translation There is another type of software that also does translation. This is called an interpreter.
- Differences between compiler and interpreter :



Compiler	Interpreter
Scans the entire program before translating it into machine code.	Translates and executes the program line by line.
Converts the entire program to machine code and executes program only when all the syntax errors are removed.	The interpreter executes one line at a time, after checking and correcting its syntax errors and then converting it to machine code.
Slow in debugging or removal of mistakes from a program.	Good for fast debugging.
Program execution time is less.	Program execution time is more.

COMPILING & EXECUTING HIGH LEVEL LANGUAGE

- The compiling process consists of two steps: the analysis of the source program and the synthesis of the object program in the machine language of the specified machine.

- The analysis phase uses the precise description of the source programming language.

- A source language is described using
  - (a) *lexical rules,*
  - *(b)syntax rules, and*
  - *(c)semantic* rules.

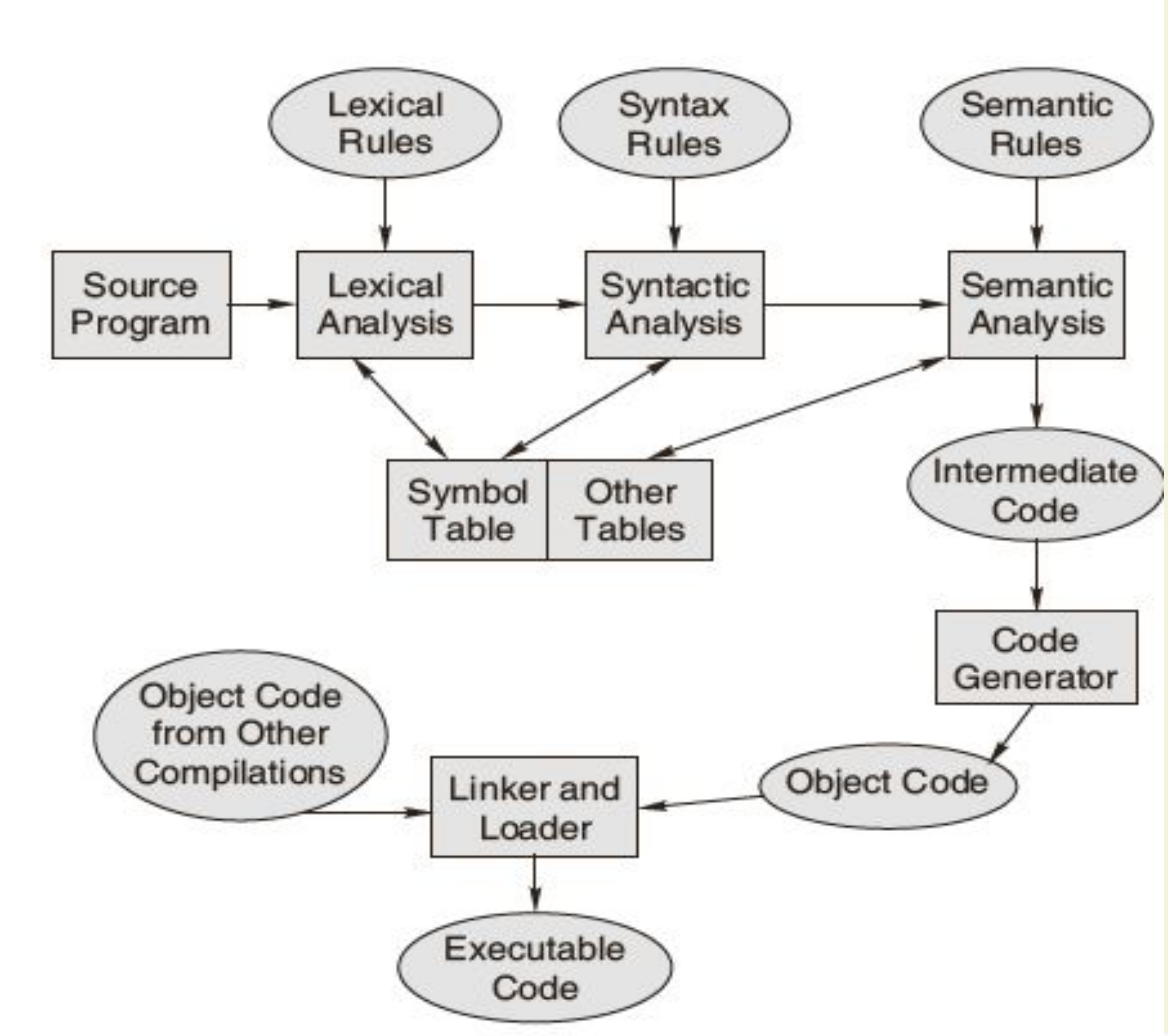
- Lexical rules specify the valid syntactic elements or words of the language.

- Syntax rules specify the way in which valid syntactic elements are combined to form the statements of the language.
- Syntax rules are often described using a notation known as BNF (Backus Naur Form) grammar.

- Semantic rules assign meanings to valid statements of the language.

The steps in the process of translating a source program in a high-level language to executable code are depicted in Fig.

THE PROCESS OF COMPILATION



THE PROCESS OF COMPILATION

- The first block is the lexical analyzer. It takes successive lines of a program and breaks them into individual lexical items namely, identifier, operator delimiter, etc. and attaches a type tag to each of these.
- Beside this, it constructs a symbol table for each identifier and finds the internal representation of each constant. The symbol table is used later to allocate memory to each variable.
- The second stage of translation is called syntax analysis or parsing.
- In this phase, expressions, declarations, and other statements are identified by using the results of lexical analysis.
- Syntax analysis is done by using techniques based on formal grammar of the programming language.
- In the semantic analysis phase, the syntactic units recognized by the syntax analyzer are processed.
- An intermediate representation of the final machine language code is produced.

THE PROCESS OF COMPILATION

- The last phase of translation is code generation, when optimization to reduce the length of machine language program is carried out.
- The output of the code generator is a machine level language program for the specified computer.
- If a subprogram library is used or if some subroutines are separately translated and compiled, a final linking and loading step is needed to produce the complete machine language program in an executable form.
- If subroutines were compiled separately, then the address allocation of the resulting machine language instructions would not be final.
- When all routines are connected and placed together in the main memory, suitable memory addresses are allocated.
- The linker's job is to find the correct main memory locations of the final executable program. The loader then places the executable program in memory at its correct address.

EXECUTION STEPS OF A PROGRAM

- Steps : the execution of a program written in high level language involves the following steps

1. ***Translation of the program resulting in the object*** program.
2. ***Linking of the translated program with other object*** programs needed for execution, thereby resulting in a binary program.
3. ***Relocation of the program to execute from the specific*** memory area allocated to it.
4. ***Loading of the program in the memory for the purpose*** of execution.

LINKER

- Linking resolves symbolic references between object programs. It makes object programs known to each other.
- Linking makes the addresses of programs known to each other so that transfer of control from one subprogram to another or a main program takes place during execution.
- In FORTRAN/COBOL , all program units are translated separately.

RELOCATION

- Relocation is more than simply moving a program from one area to another in the main memory.

- Relocation means adjustment of all address-dependent locations, such as address constant, to correspond to the allocated space, which means simple modification of the object program so that it can be loaded at an address different from the location originally specified.

LOADER

- Loading means physically placing the machine instructions and data into main memory, also known as primary storage area.
- The functions performed by the loader are :

a. Assignment of load-time storage area to the program

b. Loading of program into assigned area

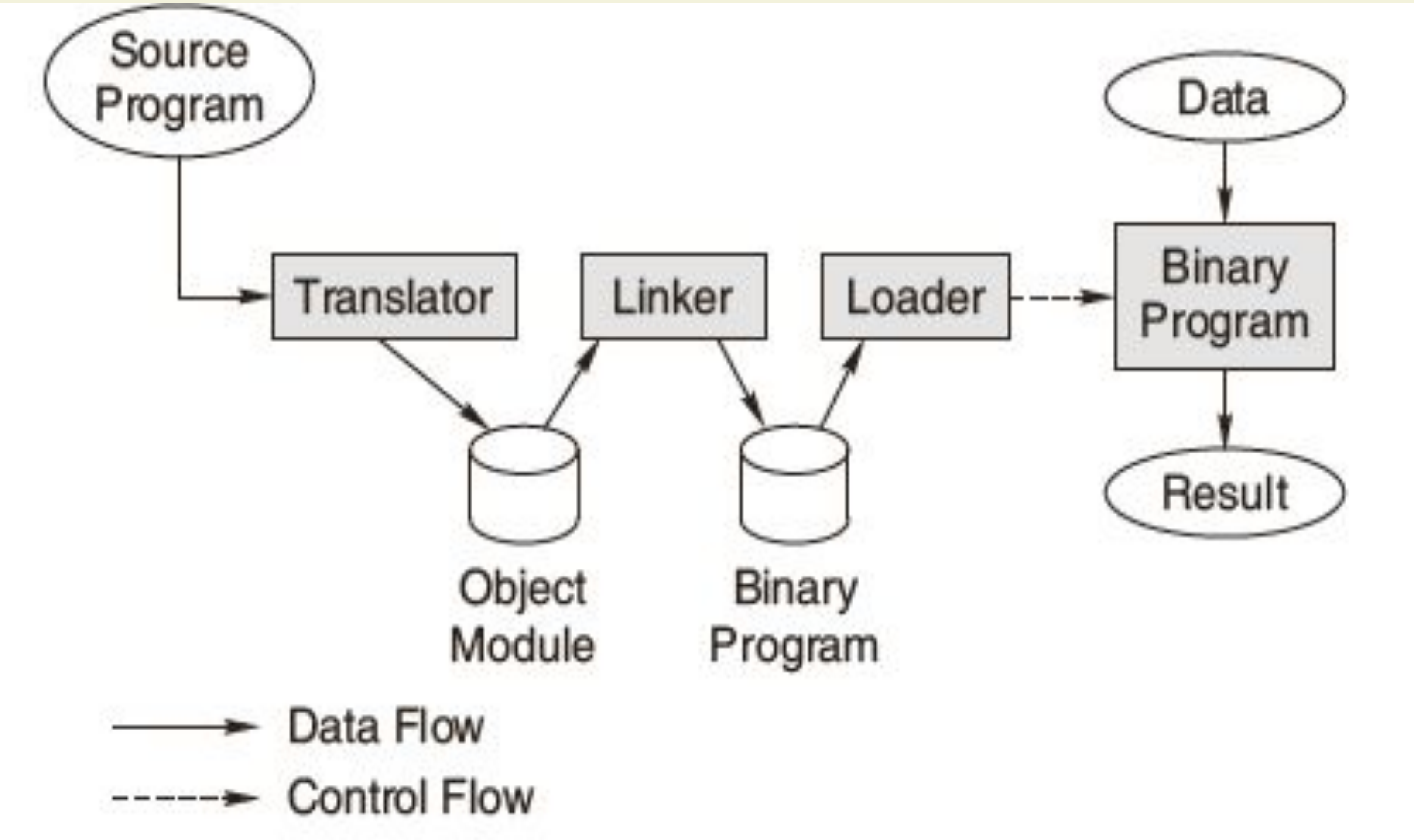
c. Relocation of program to execute properly from its load time storage area

d. Linking of programs with one another

PROGRAM EXECUTION

- When a program is compiled and linked, each instruction and each item of data is assigned an address.
- At execution time, the CPU finds instructions and data from these addresses.
- The program counter, is a CPU register that holds the address of the next instruction to be executed in a program.
- The CPU has random access capability to any and all words of the memory, no matter what their addresses.

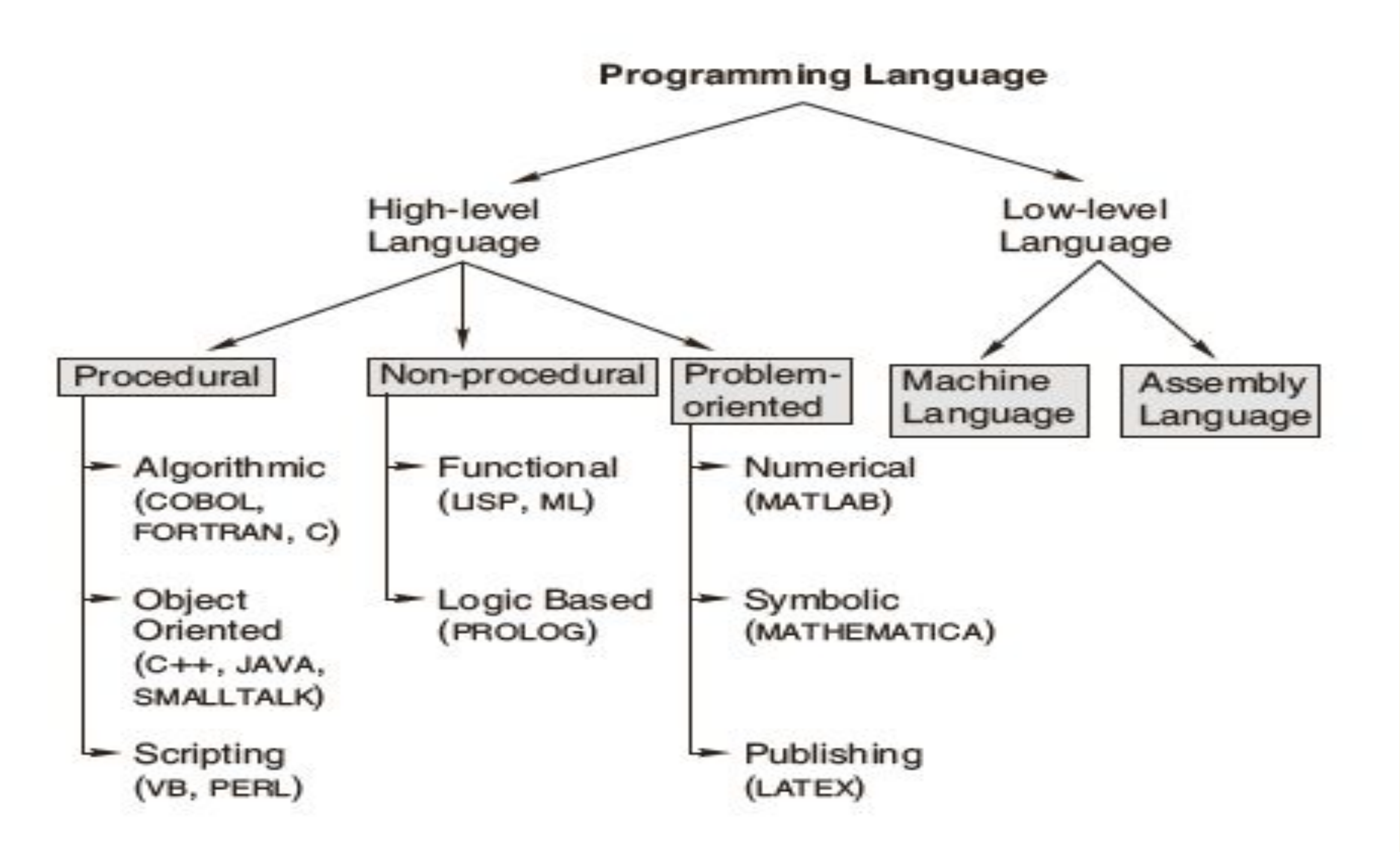
BLOCK DIAGRAM OF PROGRAM EXECUTION



THIRD,FORTH & FIFTH GENERATION LANGUAGE

- Third generation programming language specifies how to perform a task using a large number of procedural instructions and is file oriented.
- Fourth generation programming language specifies what task has to be performed using fewer instructions and is database oriented.
- Fifth generation programming language resembles human speech and eliminates the need for the user or programmer to learn a specific vocabulary, grammar ,or syntax.

CLASSIFICATION OF PROGRAMMING LANGUAGES



STUCTURED PROGRAMMING

- Structured programming involves top–down analysis for program solving, modularization of program structure and organizing structured code for individual module.
- Top-down analysis breaks the whole problem into smaller logical tasks and defines the hierarchical link between the tasks.
- Modularization of program structure means making the small logical tasks into independent program modules that carries out the desired tasks.
- Structured coding is structured programming which consists of writing a program that produces a well organized module.

ALGORITHM

- An algorithm is ‘an effective procedure for solving a problem in a finite number of steps’.
- A well-designed algorithm has termination and correctness properties.
- The four common ways of representing an algorithm are the Step-form, Pseudo-code, Flowchart and Nassi-Schneiderman .
- algorithms show these three features:

a. Sequence (also known as process)

b. Decision (also known as selection)

c. Repetition (also known as iteration or looping)

VARIABLE & SUBROUTINE

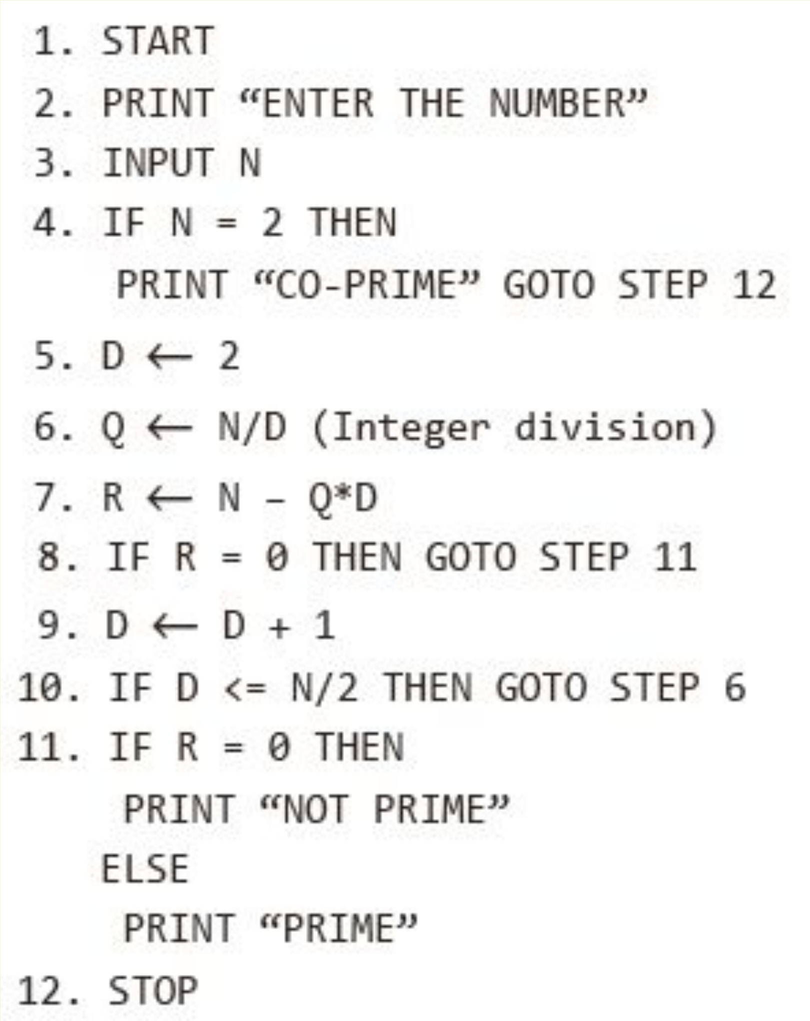
- A variable, which has a name, is a container for a value that may vary during the execution of the program.
- A subroutine is a logical collection of instructions that is invoked from within a larger program to perform a specific task.
- The subroutine is relatively independent of the remaining statements of the program that invokes it & can be invoked several times from several places during a single execution.
- After completing the specific task, a subroutine returns to the point of invocation in the larger program.

PSEUDO CODE & FLOW CHART

- Like step-form, Pseudo-code is a written statement of an algorithm using a restricted and well-defined vocabulary.
- A flowchart comprises of a set of standard shaped boxes that are interconnected by flow lines to represent an algorithm.
- There should be a logical start and stop to the flowchart.
- The usual direction of the flow of a procedure or system is from left to right or top to bottom.
- The intersection of flow lines should be avoided.
- Flowcharts facilitate communication between programmers and users.

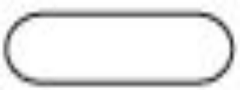
EXAMPLE: PSEUDO CODE

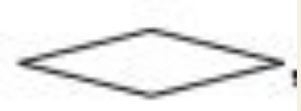
- **Problem:**
  - Write an algorithm to find out whether a given number is a prime number or not.
- **Solution:**
  - The algorithm for checking whether a given number is a prime number or not is as follows:



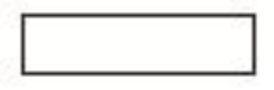
```
1. START
2. PRINT "ENTER THE NUMBER"
3. INPUT N
4. IF N = 2 THEN
     PRINT "CO-PRIME" GOTO STEP 12
5. D ← 2
6. Q ← N/D (Integer division)
7. R ← N - Q*D
8. IF R = 0 THEN GOTO STEP 11
9. D ← D + 1
10. IF D <= N/2 THEN GOTO STEP 6
11. IF R = 0 THEN
      PRINT "NOT PRIME"
    ELSE
      PRINT "PRIME"
12. STOP
```

FLOW CHARTS : SYMBOLIC REPRESENTATION

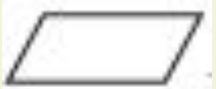
- The START and STOP are represented by an ellipse-like figure : 
- Decisions construct by the rhombus-like figure :



- The processes by rectangles :

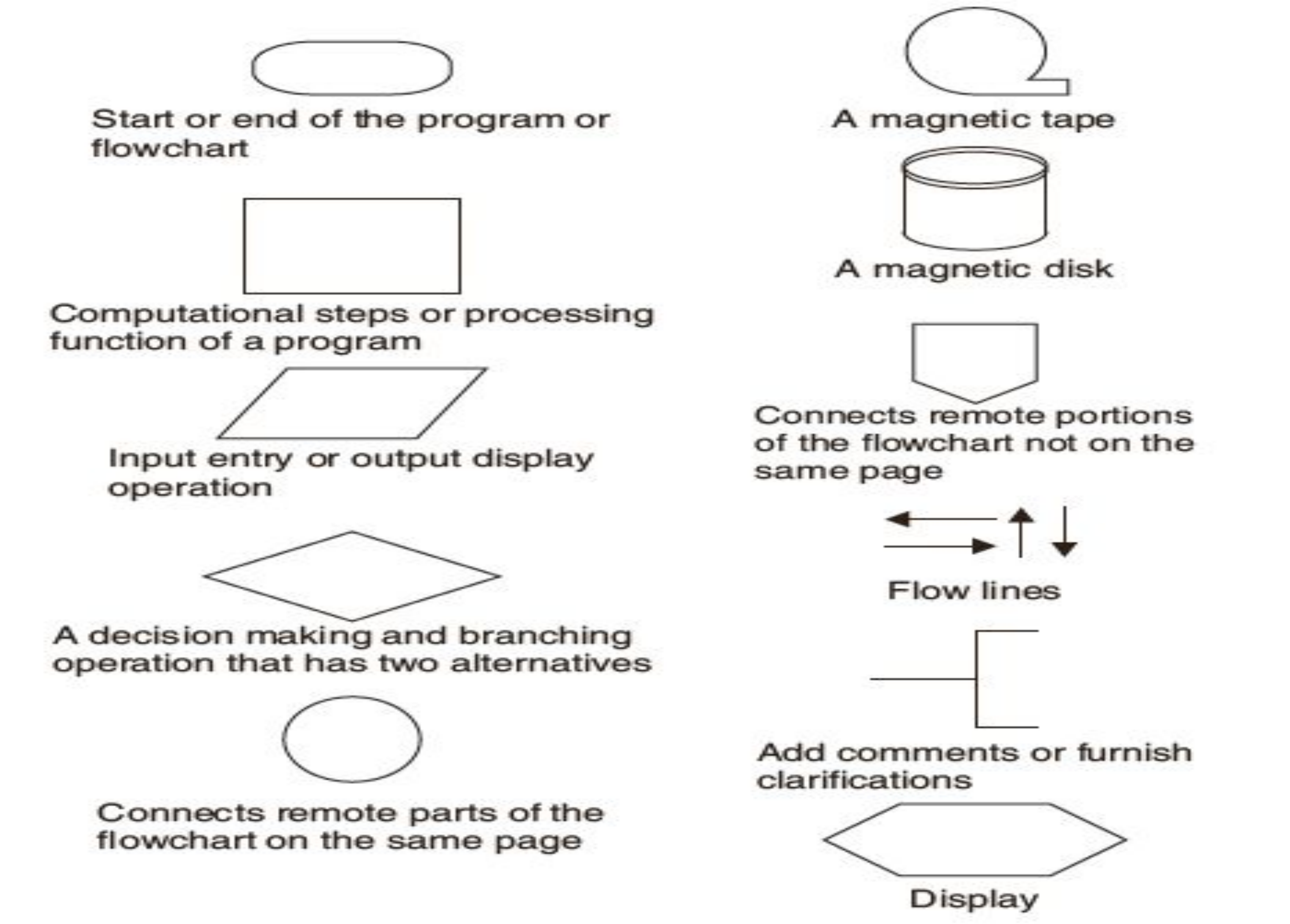


- Input / Output by parallelograms :



- Lines and arrows connect these blocks.

FLOW CHARTS:SYMBOLIC REPRESENTATION


Start or end of the program or flowchart
Computational steps or processing function of a program
Input entry or output display operation
A decision making and branching operation that has two alternatives
Connects remote parts of the flowchart on the same page
A magnetic tape
A magnetic disk
Connects remote portions of the flowchart not on the same page
Flow lines
Add comments or furnish clarifications
Display

FLOW-CHART ADVANTAGES

- *Communication*
- *Effective analysis*
- *Proper documentation*
- *Efficient coding*
- *Proper debugging*
- *Efficient program maintenance*

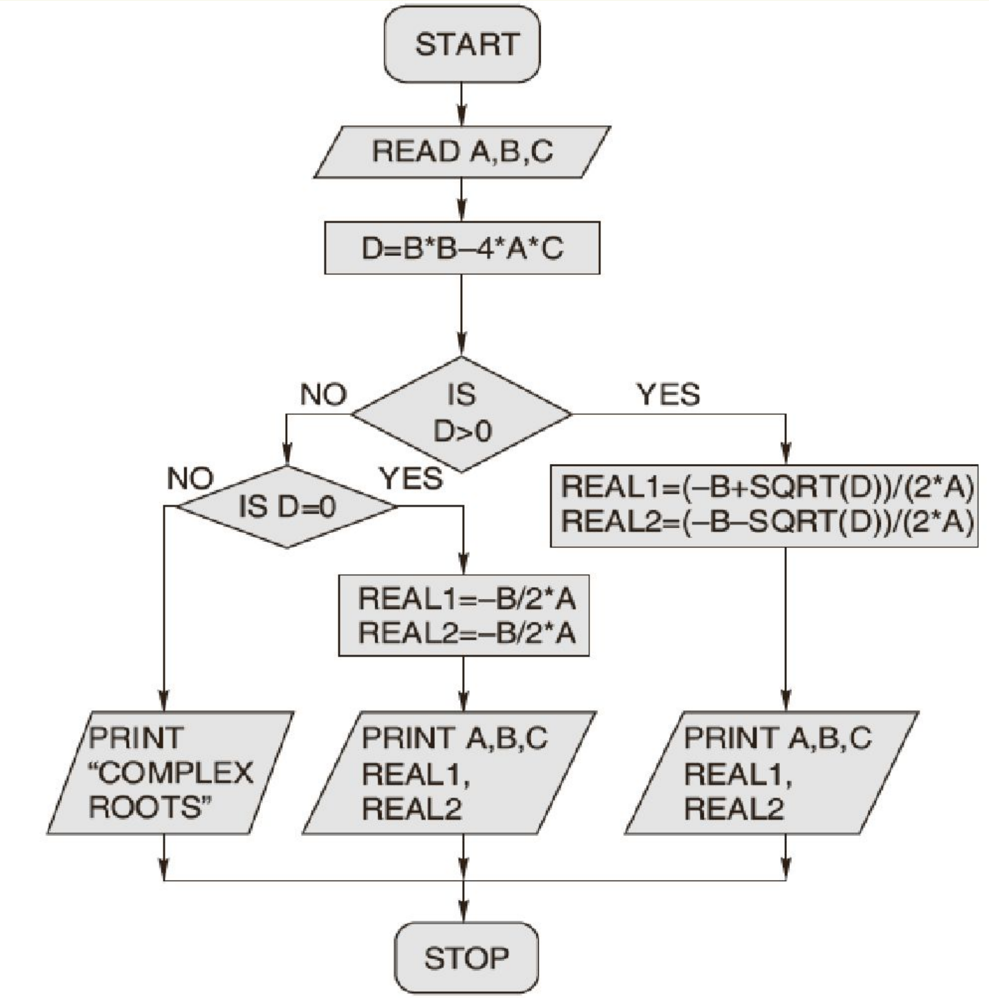
FLOW-CHARTS LIMITATIONS

- *Complex logic*
- *Alterations and modifications*
- *Reproduction*
- *Loss of objective*

FLOWCHART EXAMPLE

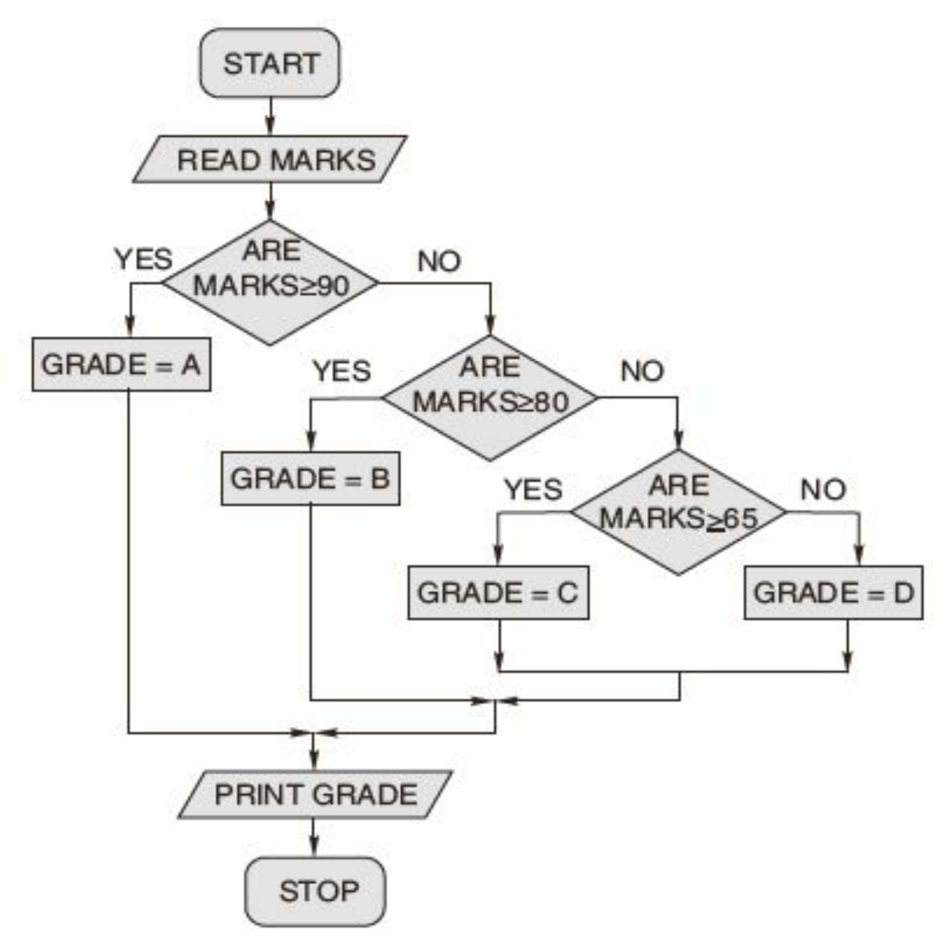
- **Problem:**

  Draw a flowchart to find the roots of a quadratic equation.



FLOW-CHART EXAMPLE

- **Problem:**
  - Prepare a flowchart to read the marks of a student and classify them into different grades. If the marks secured are greater than or equal to 90, the student is awarded Grade *A; if they are greater* than or equal to 80 but less than 90, Grade *B is awarded; if they* are greater than or equal to 65 but less than 80, Grade *C is* awarded; otherwise Grade *D is awarded.*



ALGORITHM : DESIGNING STATEGY

1. Identify the outputs needed.
2. Identify the input variables available.
3. Identify the major decisions and conditions.
4. Identify the processes required to transform inputs into required outputs.
5. Identify the environment available.

TOP DOWN DEVELOPMENT STEP

- The top-down development phase plans out the way the solution has to be done by breaking it into smaller modules and establishing a logical connection among them.
- ***Stepwise refinement :***

a. Work out each and every detail for each small piece of manageable solution procedure.

b. Decompose any solution procedure into further smaller pieces and iterate until the desired level of detail is achieved.

CONT.

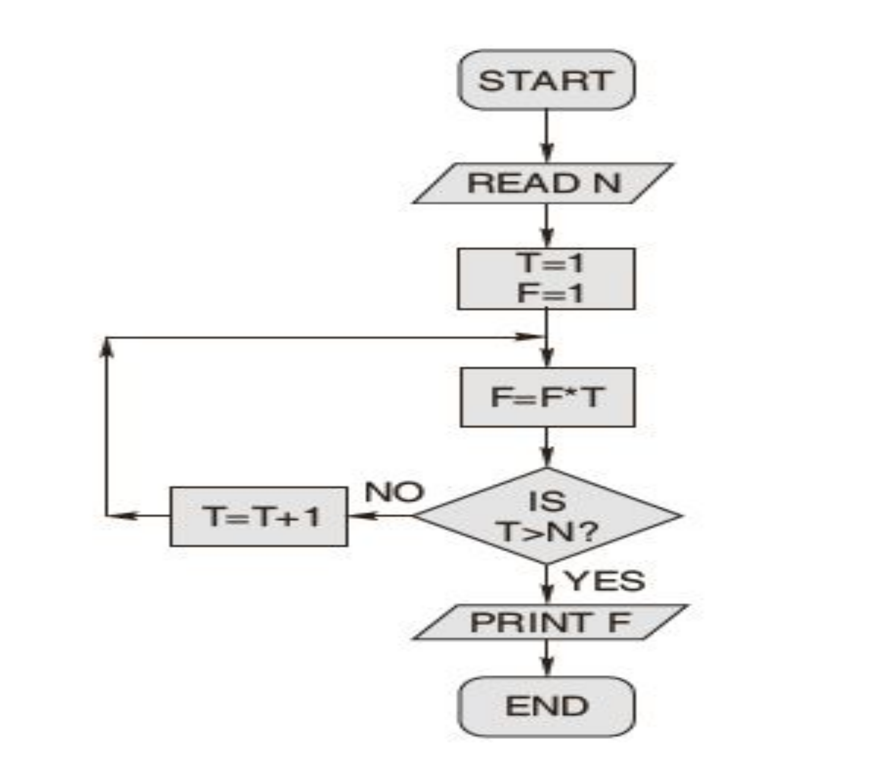
c. Group processes together which have some commonality.

d. Group variables together which have some appropriate commonality.

e. Test each small procedure for its detail and correctness and its interfacing with the other small procedures.

TRACING AN ALGORITHM TO DEPICT LOGIC

- An algorithm can be traced by verifying every procedure one by one to determine and confirm the corresponding result that is to be obtained.
- Example:



CONVERSION

- **Specification for Converting Algorithms into Programs:**

  The general procedure to convert an algorithm into a program is to code the algorithm using a suitable programming language, check the program code by employing the desk-check method and finally evaluate and modify the program, if needed.