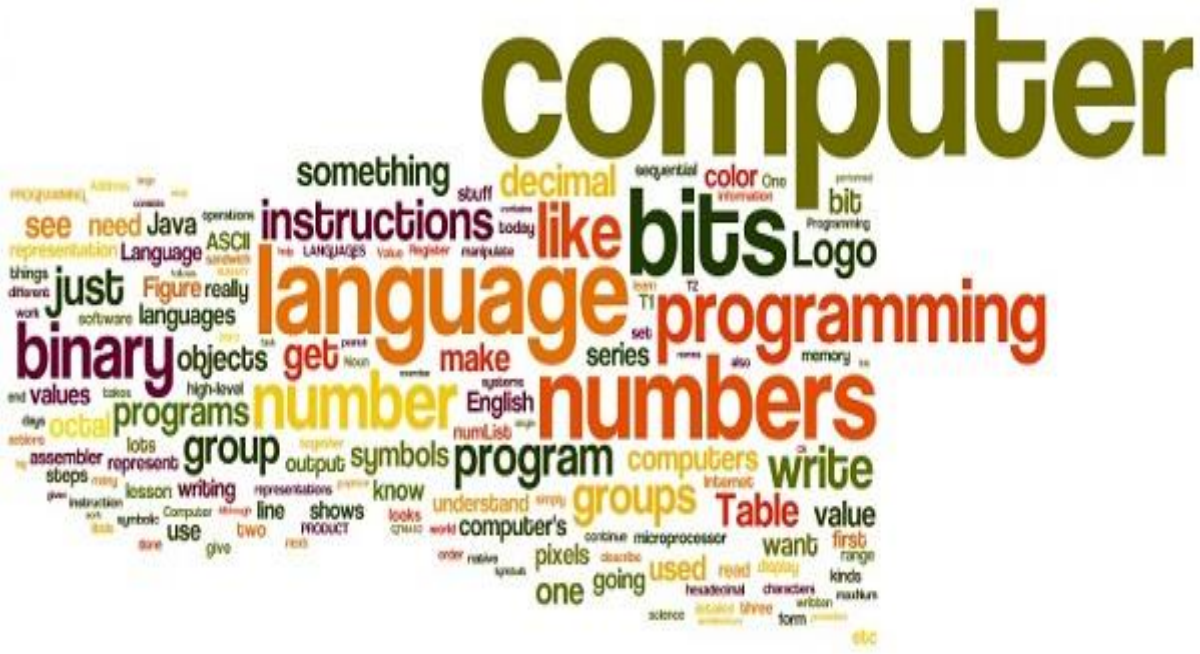


Programming Computer

Prepared by:

Dr: Eman Belal Ahmed

Lecturer in the Electrical Engineering Department

List of Contents

CHAPTER 1: Software Design.

- **Introduction to software design**
- **Evolution and comparison of programming languages**
- **Language translators, Types of Translators**
- **Number system, Number system conversion**
- **General questions of chapter 1**

CHAPTER 2: Structured Function

- **Basics of Structured Function**
- **Object oriented and multi-core programming.**
- **General questions of chapter 2**

CHAPTER 3: parallel programming& introduction to MATLAB

- **Introduction to parallel programming**
- **Program maintenance & testing**
- **MATLAB basics**
- **Algorithm and flowchart**

CHAPTER 4: Documentation

- **Documentation**
- **Numerical and non-numerical examples**
- **Programming project**
- **General questions of chapter 3, 4**

References

CHAPTER (1)

Software Design

CHAPTER 1:Software Design

1.1 ***What is the computer:***

A computer is an electronic device, operating under the control of instructions stored in its own memory that can accept data (input), process the data according to specified rules, produce information (output), and store the information for future use.

A process has steps for transforming inputs (data) to output (information). **Computer programming** is the process of writing code to facilitate specific actions in a computer, application or software program, and instructs them on how to perform. Computer programmers are professionals that create instructions for a computer to execute by writing and testing code that enables applications and software programs to operate successfully.

1.2 ***Functionalities of a computer:***

Any digital computer carries out five functions in gross terms:

a) Takes data as input.
b) Stores the data/instructions in its memory and use them when required.
c) Processes the data and converts it into useful information as shown in Fig. 1.
d) Generates the output.
e) Controls all the above four steps.



Fig. 1: Function of a computer.

1.3 ***Characteristics of computers:***

Computer has the following characteristics

1) Accuracy

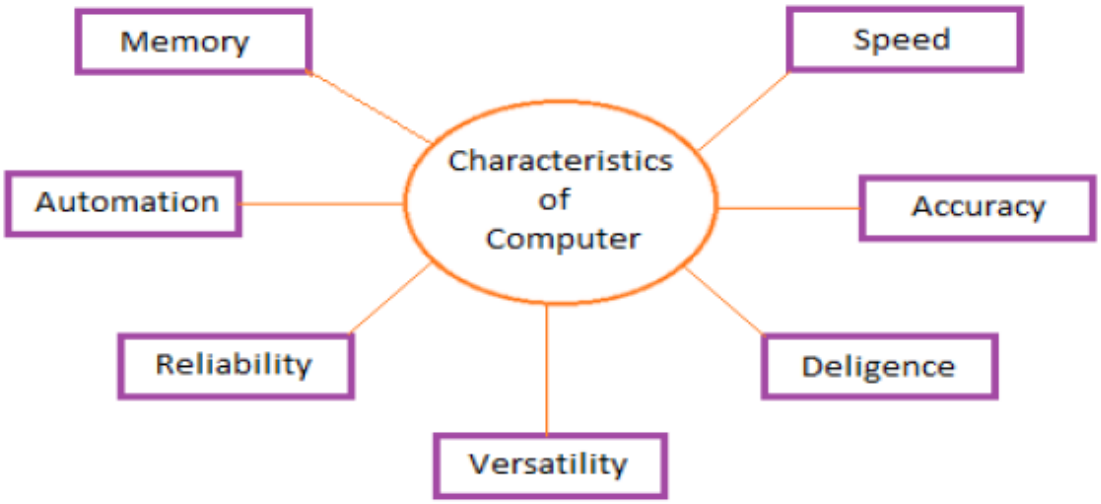
2) Diligence

3) Versatility

4) Reliability

5) Automation

6) Storage



1) Speed

- A computer works with much higher speed compared to humans
- Computers can process millions of instructions per second.
- The time taken by computers for their operations is microseconds and nanoseconds.

2) Accuracy

- Computers perform calculations with 100% accuracy.
- Errors that may occur can almost always be attributed to human error.
- Inaccurate data, poorly designed system or faulty programs written by the programmer

3) Diligence

- Computers are highly consistent.
- They do not suffer from human traits of boredom and tiredness, lack of concentration.
- Computers can perform voluminous and repetitive jobs.

4) Versatility

- Computers are versatile machines and are capable of performing any task as long as it
- Can be broken down into a series of logical steps.
- Computer to perform different kinds of works with same accuracy and efficiency.

5) Reliability

- A computer is reliable as it gives consistent result for similar set of data
- We give same set of input any number of times; we will get the same result.

6) Automation

- Computer performs all the tasks automatically. It performs tasks without manual intervention.

7) Memory/Storage

- Computers can store large volumes of data.
- A piece of information once recorded (or stored) in the computer, can never be forgotten and can be retrieved almost instantaneously.

1.4 ***Computer components:***

Any kind of computers consists of hardware and software:

A- Hardware:

Computer hardware as shown in Fig. 2 is the collection of physical elements that constitutes a computer system. Computer hardware refers to the physical

parts or components of a computer such as the monitor, mouse, keyboard, computer data storage, hard drive disk (HDD), system unit (graphic cards, sound cards, memory, motherboard and chips), etc. all of which are physical objects that can be touched.

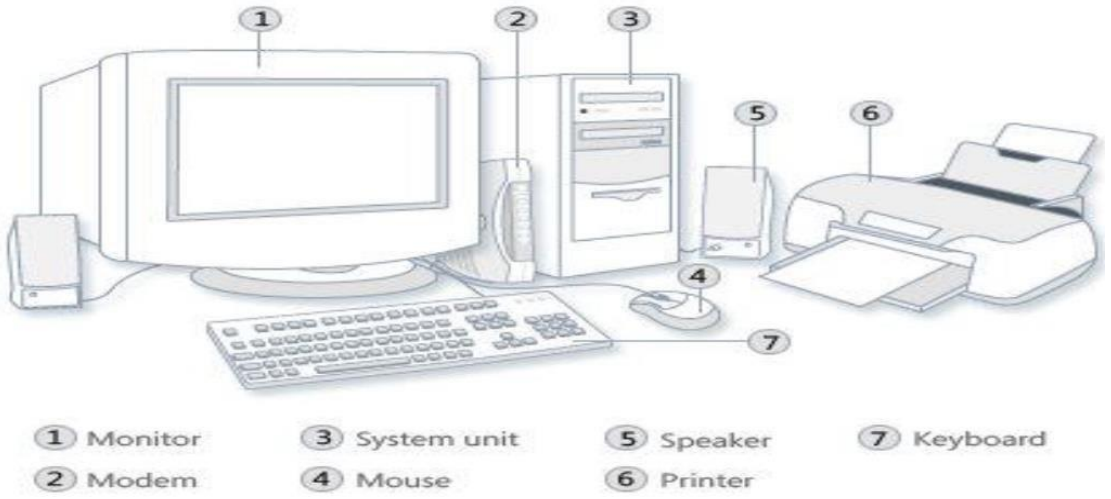


Fig. 2: Hardware components of a computer.

❖ ***Input Devices:***

Input device as shown in Fig. 3 is any peripheral (piece of computer hardware equipment to provide data and control signals to an information processing system such as a computer or other information appliance. Input device Translate data from form that humans understand to one that the computer can work with. Most common are keyboard and mouse.

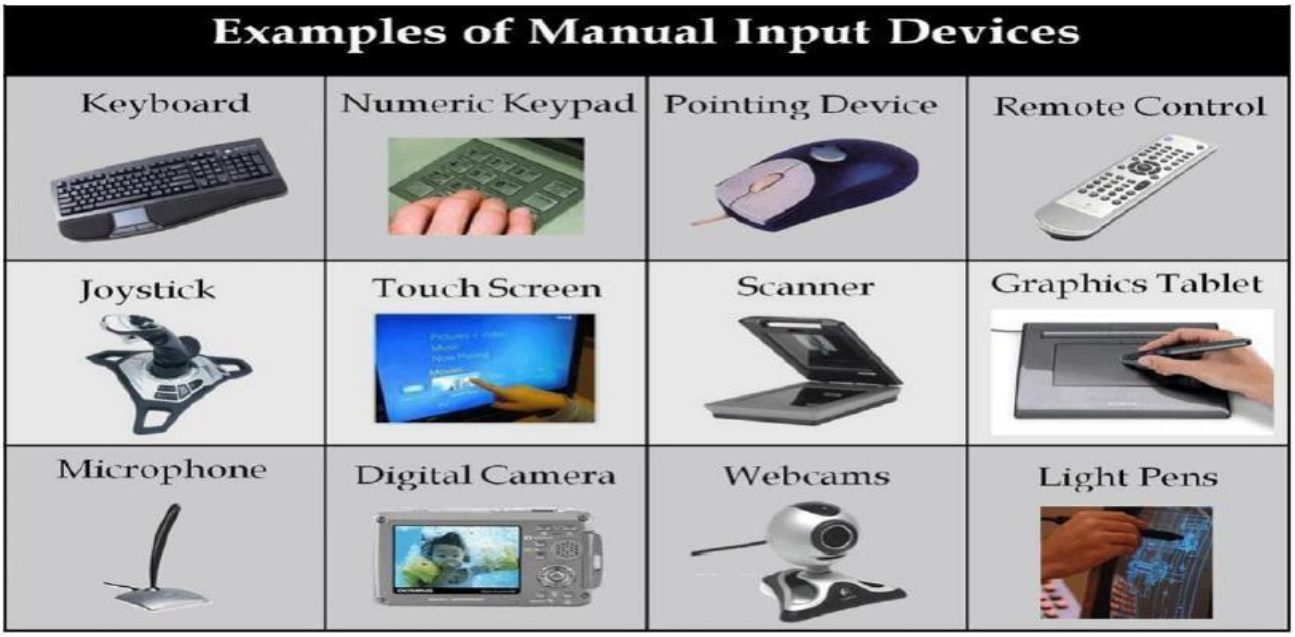


Fig. 3: Input devices of a computer.

❖ ***Central Processing Unit (CPU):***

A CPU as shown in Fig. 4 is brain of a computer. It is responsible for all functions and processes. Regarding computing power, the CPU is the most important element of a computer system.

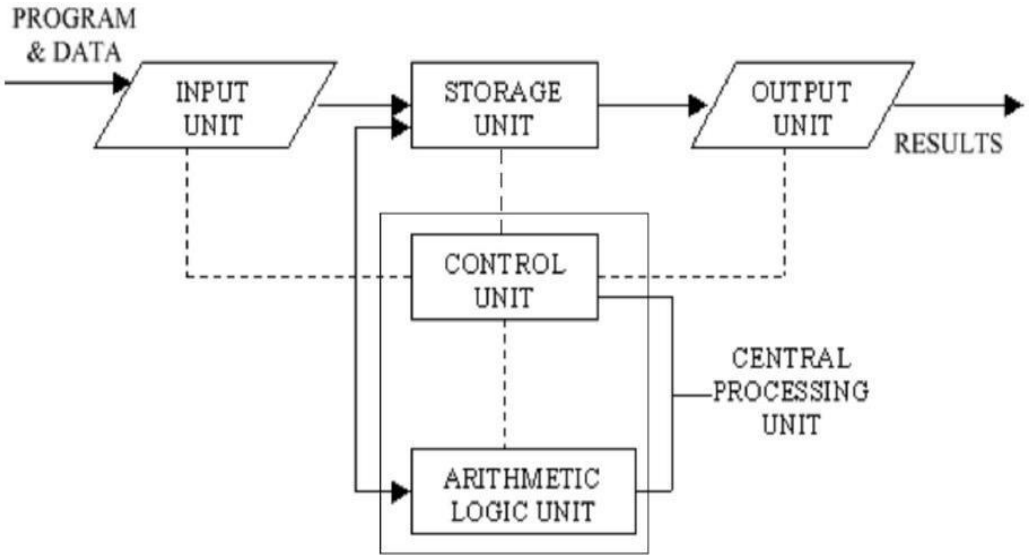


Fig. 4: CPU of a computer.

❖ **The CPU is comprised of three main parts:**

a. **Arithmetic Logic Unit (ALU):** Executes all arithmetic and logical operations. Arithmetic calculations like as addition, subtraction, multiplication and division. Logical operation like compare numbers, letters, or special characters.

b. **Control Unit (CU): controls and co-ordinates computer components.**

- Read the code for the next instruction to be executed.
- Increment the program counter so it points to the next instruction.
- Read whatever data the instruction requires from cells in memory.
- Provide the necessary data to an ALU or register.
- If the instruction requires an ALU or specialized hardware to complete, instruct the hardware to perform the requested operation.

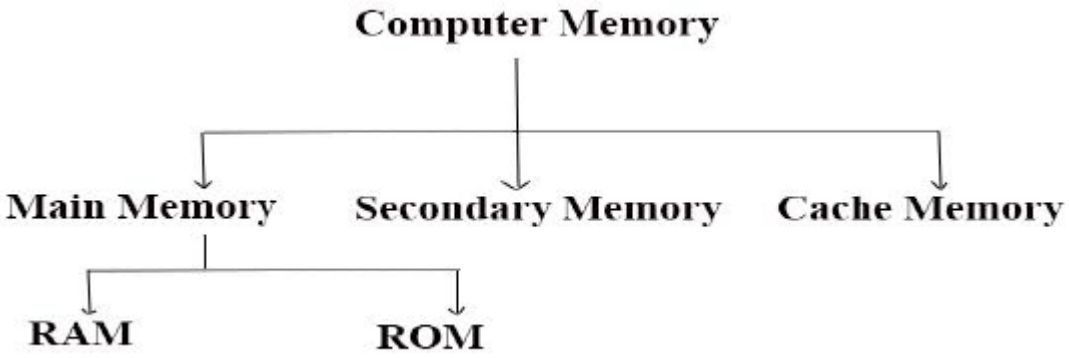
c. **Registers:** Stores the data that is to be executed next, "very fast storage area".

1.5 *COMPUTER MEMORY:*

A computer memory is just like a human brain. It is used to store data and instructions. Computer memory is the storage space in computer where data is to be processed and instructions required for processing are stored. Memory is divided into large number of small parts called cells. Each location or cell has a unique address .Computer storage can be divided into primary memory, secondary memory and cache memory.

Computer Memory are three types :

1. Primary Memory/Main Memory
2. Secondary Memory
3. Cache Memory



1- Primary Memory:

i. **RAM:** **R**andom **A**ccess **M**emory (RAM) is a memory scheme within the computer system responsible for storing data on a temporary basis, so that it can be promptly accessed by the processor as and when needed. It is volatile in nature, which means that data will be erased once supply to the storage device is turned off. RAM stores data randomly and the processor accesses these data randomly from the RAM storage. RAM is considered "random access" because you can access any memory cell directly if you know the row and column that intersect at that cell.

ii. **ROM:** **R**ead **O**nly **M**emory ROM is a permanent form of storage. ROM stays active regardless of whether power supply to it is turned on or off. ROM devices do not allow data stored on them to be modified.

Comparison between ROM and RAM

ROM	RAM
It stands *Read-Only Memory*.	**It stands for *Random Access Memory*.**
It is a Read-Only Memory i.e. the data can be only accessed and cannot be re-write or altered by the CPU.	It is a Read-Write memory i.e. the data can be stored and accessed by the CPU.

It ***permanently stores*** the data.	It ***temporarily stores*** the data.
It stores the program code or instructions needed at the boot-up or start-up of the system.	It stores the data that is currently being used by the CPU.
It is ***a non-volatile memory*** i.e. the data kept stored regardless of the removal of power supply.	It is ***a volatile memory*** i.e. the data is erased when the power supply is removed.
It does not need a power supply to retain its data.	It ***needs a constant*** supply of power to retain its information.
The ROM is cheaper than RAM.	***The RAM is comparatively expensive than ROM.***
The ROMs have very low capacity measured in ***MegaBytes***.	The RAM is available in fairly huge storage capacity measured in ***GigaBytes.***

2- Secondary Memory:

Stores data and programs permanently: it's retained after the power is turned off

i. **Hard drive (HD):** A hard disk is part of a unit, often called a "disk drive," "hard drive," or "hard disk drive," that store and provides relatively quick access to large amounts of data on an electromagnetically charged surface or set of surfaces.





ii. **Optical Disk:** an optical disc drive (ODD) is a disk drive that uses laser light as part of the process of reading or writing data to or from optical discs. Some drives can only read from discs, but recent drives are commonly both readers and recorders, also called burners or writers. Compact discs, DVDs, and Blu-ray discs are common types of optical media which can be read and recorded by such drives. Optical drive is the generic name; drives are usually described as "CD" "DVD", or "Bluray", followed by "drive", "writer", etc. There are three main types of optical media: CD, DVD, and Blu-ray disc. CDs can store up to 700 megabytes (MB) of data and DVDs can store up to 8.4 GB of data. Blu-ray discs, which are the newest type of optical media, can store up to 50 GB of data. This storage capacity is a clear advantage over the floppy disk storage media (a magnetic media), which only has a capacity of 1.44 MB.



iii. **Flash Disk:** A storage module made of flash memory chips. A Flash disks have no mechanical platters or access arms, but the term "disk" is used because the data are accessed as if they were on a hard drive. The disk storage structure is emulated.

❖ **Output devices:**

An output device as illustrated in Fig. 5 is any piece of computer hardware equipment used to communicate the results of data processing carried out by an information processing system (such as a computer) which converts the

electronically generated information into human-readable form.

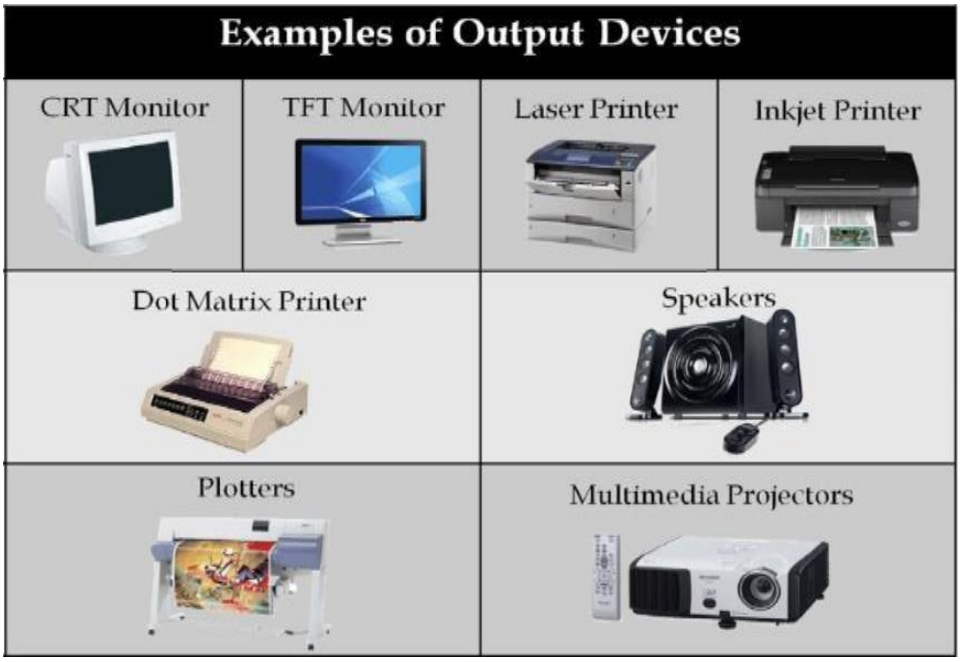


Fig. 5: Output devices of a computer.

B- Software:

Software is a generic term for organized collections of computer data and instructions, often broken into two major categories: system software and application software.

Software types:

1) System software:

is responsible for controlling, integrating, and managing the individual hardware components of a computer system so that other software and the users of the system see it as a functional unit without having to be concerned with the low-level details such as transferring data from memory to disk, or rendering text onto a display

Generally, system software consists of an operating system and some fundamental utilities such as disk formatters, file managers, display managers,

text editors, user authentication (login) and management tools, and networking and device control software.



2) Application software:

is used to accomplish specific tasks other than just running the computer system. Application software may consist of a single program, such as an image viewer; a small collection of programs (often called a software package) that work closely together to accomplish a task, such as a spreadsheet or text processing system; a larger collection (often called a software suite) of related but independent programs and packages that have a common user interface or shared data format, such as Microsoft Office, which consists of closely integrated word processor, spreadsheet, database, etc.; or a software system, such as a database management system, which is a collection of fundamental programs that may provide some service to a variety of other independent applications.



Application software vs system software

Table 1.1: Comparison between application software and system software

APPLICATION SOFTWARE	SYSTEM SOFTWARE
It performs a specific task	It operates the computer hardware
It runs when the user requests it	It runs automatically when the computer starts and stops when the computer is turned off
It cannot run without system software support	It gives a platform for application software
Application software is installed by the user	System software comes bundled with the operating system
The user interacts with application software	Users cannot interact directly with system software and it runs in the background
It cannot run independently and depends upon system software	It can run independently

Examples are ***MS Office, Google Chrome, Firefox, Adobe Photoshop, Calculator, etc***	Examples are ***Windows OS, Mac OS, Android OS, Apple iOS, Linux OS***

1.6 ***Computers classification:***

Computers can be generally classified by size and power as follows, though there is Considerable overlap:

1. **Personal computer:** A small, single-user computer based on a microprocessor. In addition to the microprocessor, a personal computer has a keyboard for entering data, a monitor for displaying information, and a storage device for saving data.



2. **Workstation:** A powerful, single-user computer. A workstation is like a personal computer, but it has a more powerful microprocessor and a higher-quality monitor.



3. **Minicomputer:** A multi-user computer capable of supporting from 10 to hundreds of users simultaneously.



4. **Mainframe:** A powerful multi-user computer capable of supporting many hundreds or thousands of users simultaneously.



5. **Supercomputer:** An extremely fast computer that can perform hundreds of millions of instructions per second.



1.7 ***Application of computers***

The computer used in various application area in the world.

1. Business
2. Education
3. Marketing
4. Banking
5. Insurance
6. Communication
7. Health Care
8. Military
9. Engineering Design

1.8 ***Data, Information and Knowledge differences :***

1) **Data:** Facts and figures which relay something specific, but which are not organized in any way and which provide no further information regarding patterns, context, etc. So data means "unstructured facts and figures that have the least impact on the typical manager".

2) **Information:** For data to become information, it must be contextualized, categorized, calculated and condensed. Information thus paints a bigger picture; it is data with relevance and purpose. It may convey a trend in the environment, or perhaps indicate a pattern of sales for a given period of time. Essentially information is found "in answers to questions that begin with such words as who, what, where, when, and how many".

3) **Knowledge:** Knowledge is closely linked to doing and implies know-how and understanding. The knowledge possessed by each individual is a

product of his experience, and encompasses the norms by which he evaluates new inputs from his surroundings.

1.9 ***Unit of Measurements:***

1) **Storage measurements:** The basic unit used in computer data storage is called a bit (binary digit) as illustrated in Table 1.2. Computers use these little bits, which are composed of ones and zeros, to do things and talk to other computers. All your files, for instance, are kept in the computer as binary files and translated into words and pictures by the software (which is also ones and zeros). This two number system, is called a "binary number system" since it has only two numbers in it. The decimal number system in contrast has ten unique digits, zero through nine.

Bit	BIT	0 or 1
Kilobyte	KB	1024 bytes
Megabyte	MB	1024 kilobytes
Gigabyte	GB	1024 megabytes
Terabyte	TB	1024 gigabytes

Table 1.2: Computer storage units.

2) Speed measurement: The speed of Central Processing Unit (CPU) is measured by Hertz (Hz) as illustrated in Table 1.3, which represent a CPU cycle. The speed of CPU is known as Computer Speed.

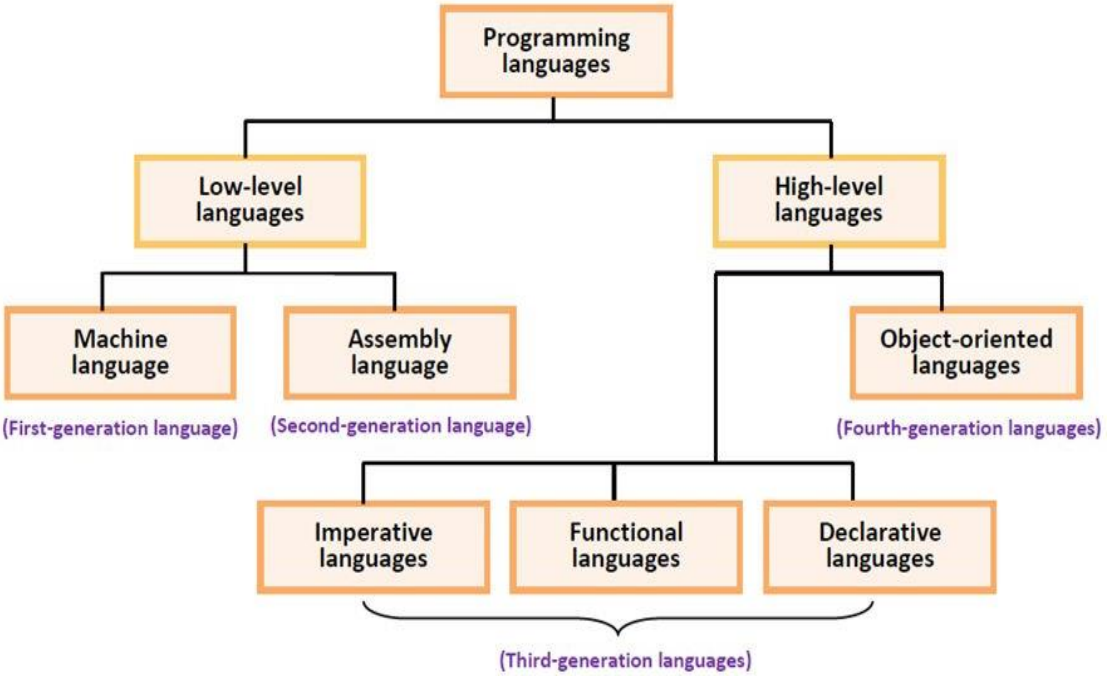
CPU SPEED MEASURES	
1 hertz or Hz	1 cycle per second
1 MHz	1 million cycles per second or 1000 Hz
1 GHz	1 billion cycles per second or 1000 MHz

Table 1.3: CPU speed measures.

1.10 ***Computer Language:***

Computer language or **programming language** is a coded syntax used by computer programmers to communicate with a computer. A language consists of set of instructions to perform a specific task to get the required output is known as **programming language**. As this language enables a computer user to dictate what commands the computer must perform to process data. These languages can be classified into following categories:

1) Machine language (Low-level language).
2) Assembly language (Low-level language).
3) High level language.



Low level languages are closer to the language used by a computer, while high-level languages are closer to human languages.

1) **Machine Language:" First Generation Language"**

❖ Based on binary language; every instruction and data should be written

using 0’s and 1’sMachine language or machine code is the native language directly understood by the computer's central processing unit or CPU. This type of computer language is not easy to understand, as it only uses a binary system, an element of notations containing only a series of numbers consisting of one and zero, to produce commands. Machine language is a collection of binary digits or bits that the computer reads and interprets.

- ❖ Machine languages are the only languages understood by computers.
- ❖ While easily understood by computers, machine languages are almost impossible for humans to use because they consist entirely of numbers.
- ❖ Different types of hardware use different machine code. For example, IBM computers use different machine language than Apple computers.

Advantages of Machine Language	**Disadvantages of Machine Language**
• Translation Free • High Speed	• Machine Dependent • Complex Language • Error Prone • Tedious, boring

2) <u>Assembly Level Language"</u> second-generation languages"

- ❖ Assembly Level Language is a set of codes that can run directly on the computer's processor. This type of language is most appropriate in writing operating systems and maintaining desktop applications. With the assembly level language, it is easier for a programmer to define commands. It is easier to understand and use as compared to machine language.
- ❖ Assembly languages are only somewhat easier to work with than machine languages. To create programs in assembly language, developers use cryptic English-like phrases to represent strings of numbers. The code is then translated into object code, using a translator called an assembler.

❖ Assembly languages use keywords and symbols, much like English, to form a programming language but at the same time introduce a new problem.

❖ The problem is that the computer doesn't understand the assembly code, so we need a way to convert it to machine code, which the computer does understand.

❖ Assembly language programs are translated into machine language by a program called an assembler.

Advantages of Assembly Language	Disadvantages of Assembly Language
• Easy to understand and use • Less Error Prone • Efficiency • More control on Hardware	• Machine Dependent • Harder to Learn • Slow Development Time • Less Efficient • No Standardization

3) High Level Language:

❖ High Level Languages are user-friendly languages which are similar to English with vocabulary of words and symbols. These are easier to learn and require less time to write. They are problem oriented rather than "machine" based. Program written in a high-level language can be translated into many machine languages and therefore can run on any computer for which there exists an appropriate translator.

❖ High-level languages allow us to write computer code using instructions resembling everyday spoken language (for example: print, if, while) which are then translated into machine language to be executed.

❖ Programs written in a high-level language need to be translated into machine

language before they can be executed.

❖ Some programming languages use a compiler to perform this translation and others use an interpreter.

❖ Higher-level programming languages are divided into three "generations," each more powerful than the last: **Third-generation languages, Fourth-generation languages, Fifth-generation languages**

Examples of High-level Language:

- ADA
- C
- C++
- JAVA
- BASIC
- COBOL
- PASCAL
- PHYTON

Advantages of HLL	Disadvantages of HLL
• Readability • Machine Independent • Easy debugging • Easier to maintain • Easy Documentation	• Poor control on Hardware • Less Efficient

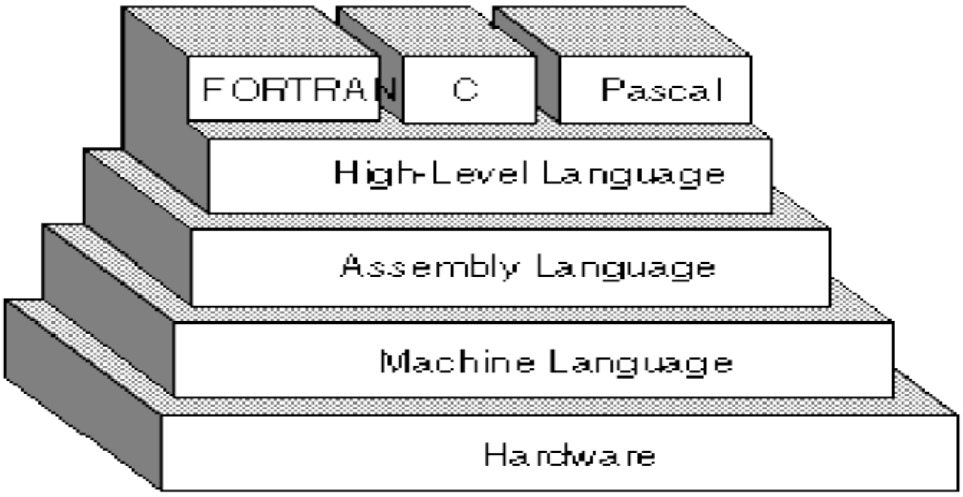


Fig. 6: Computer languages.

1.11 ***Program translators (Language Processors)***

Language Processors

Compilers, interpreters, translate programs written in high-level languages into machine code that a computer understands. And assemblers translate programs written in low-level or assembly language into machine code. In the compilation process, there are several stages. To help programmers write error-free code, tools are available.

- ***Types of Translators***

☒ ***Assembler***

☒ ***Compiler***

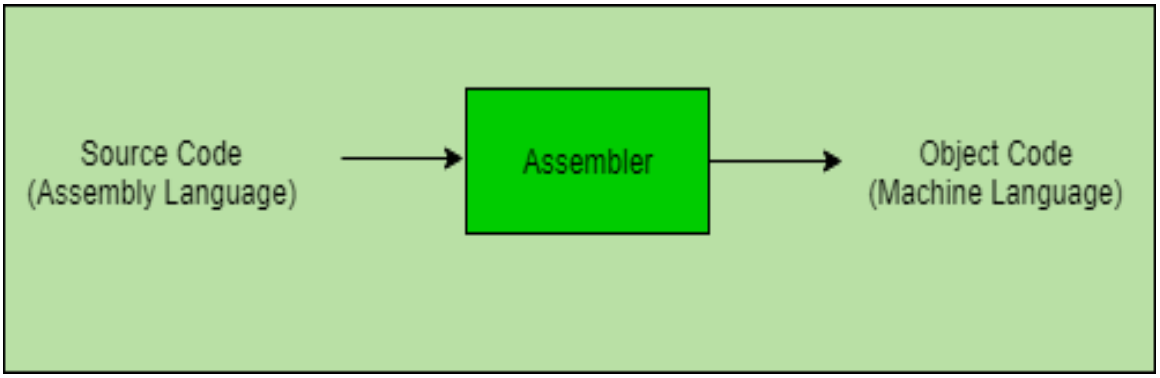
☒ ***Interpreter***

A) Assembler

The Assembler is used to translate the program written in Assembly language into machine code. The source program is an input of an assembler that contains assembly language instructions. The output generated by the assembler is the object code or machine code understandable by the computer.

Assembler is basically the 1st interface that is able to communicate humans with the machine. We need an Assembler to fill the gap between human and machine so that they can communicate with each other. code written in assembly language is some sort of mnemonics(instructions) like ADD, MUL, MUX, SUB, DIV, MOV and so on. and the assembler is basically able to convert these mnemonics in Binary code. Here, these mnemonics also depend upon the architecture of the machine.

For example, the architecture of intel 8085 and intel 8086 are different.

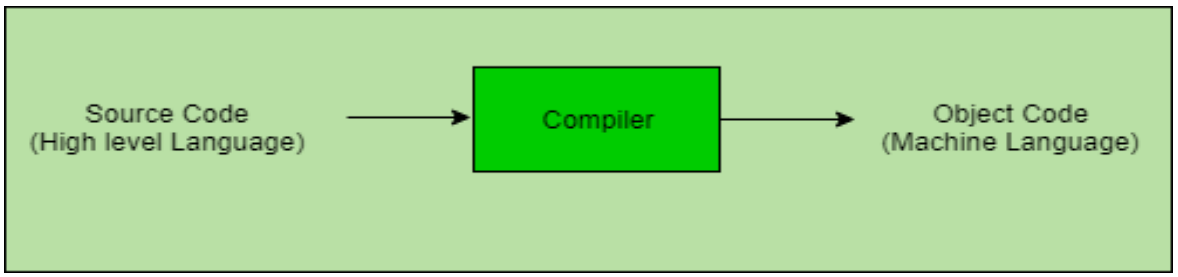


B) Compiler:

A compiler is a program translator that translates a program written in high-level language into machine language program. During the translation process, the compiler reads the source program and checks for syntax errors. In case of any errors, the compiler will not create the object code until all the errors are rectified.

Once the program has been compiled, the resulting m/c code is saved separately, and can be run on its own at any time. But, if the source code is modified then it is necessary to recompile the program again to reflect the changes. A compiler as shown in figure below is a special program that processes statements written in a particular programming language called as source code and converts them into machine language or "machine code" that a computer's processor uses. Compiler translates high level language programs directly into machine language program. This process is called **compilation**.

Example: C, C++, C#, Java.



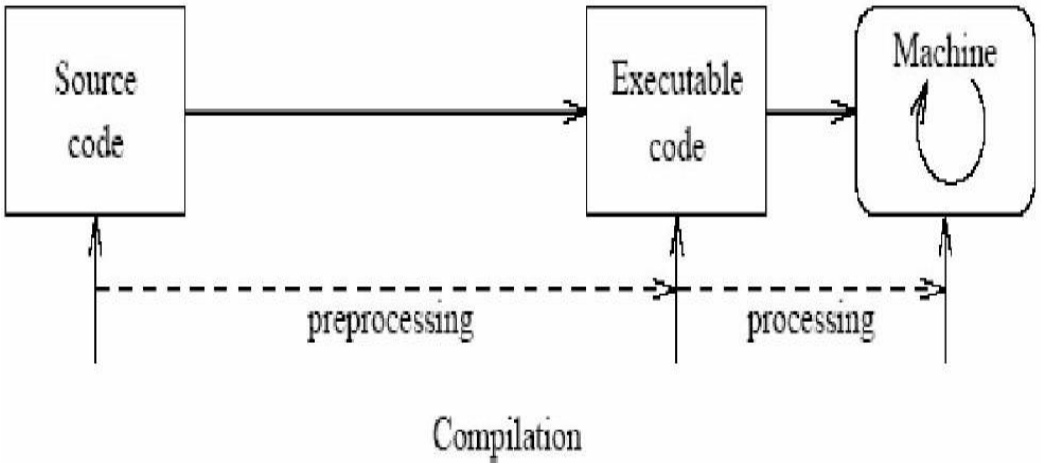


Fig. 7: Compilation process.

C. Interpreter:

Interpreter is Also a language translator. Translates high-level language into machine language. Unlike compilers, it translates a statement in a program and execute immediately, i.e., before translating the next source language statement. When an error is encountered, the execution of the program is halted and an error message is displayed. Languages such as BASIC and LISP has its own interpreters

An interpreter translates high-level instructions into an intermediate form as shown in figure below, which it then executes. Compiled programs generally run faster than interpreted programs. The advantage of an interpreter, however, is that it does not need to go through the compilation stage during which machine instructions are generated. This process can be time-consuming if the program is long.

Example: Perl, Python and Matlab.

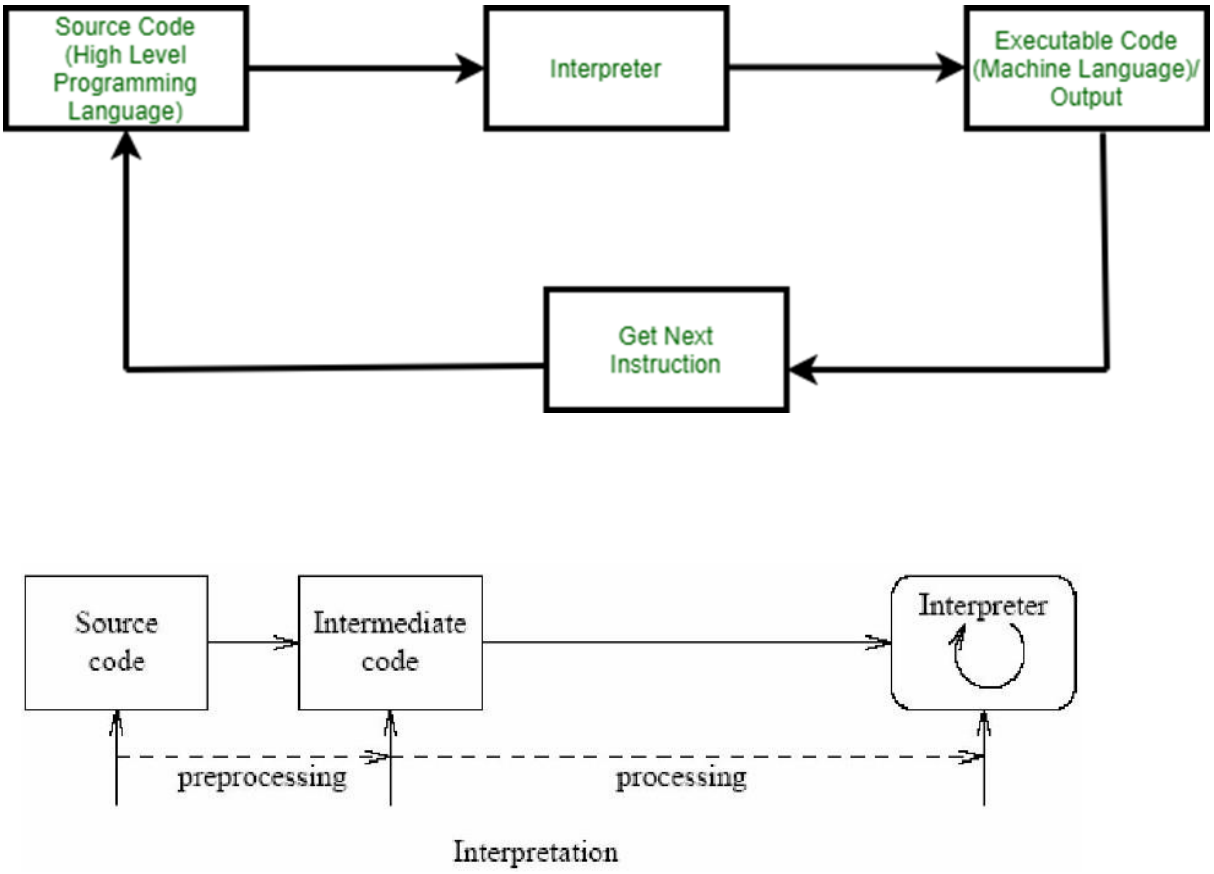
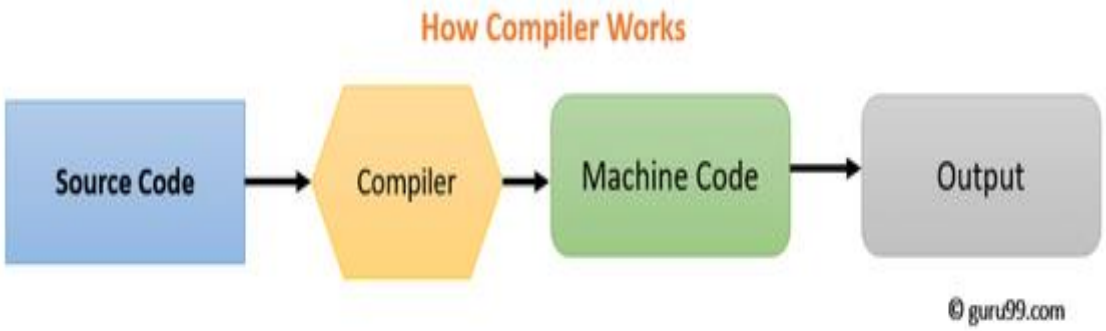
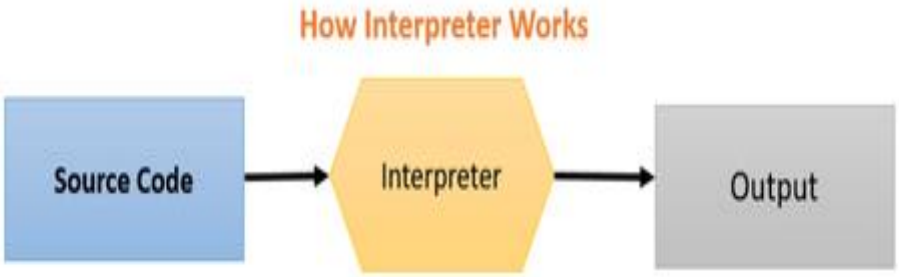


Fig. 8: Interpretation process.

Interpreter	Compiler
• Only one statement of the program is translated into machine code by the interpreter.	• Compiler scans the entire program and simultaneously translates it into a machine code.
• The interpreter checks the keywords of the program.	• The compiler checks the program syntax.
• It takes less time to analyzing and processing the source code. The time to execute the process is, however, much slower.	• It takes a long time to analyzing and processing the source code. The overall time required to complete the process is, however, much faster.
• An intermediary code is not generated by an interpreter. An interpreter is therefore highly efficient in memory.	• An intermediate object code is always generated by a compiler. It will need to be linked further. More memory is therefore required.

• Continuously translates the program until the first error is encountered. If an error is detected, it stops working and therefore debugging is easy.	• A compiler generates the error message only after scanning the entire program and therefore debugging while working with a compiler becomes relatively difficult.
• Interpreters are used in programming languages such as Ruby and Python.	• Compilers are used in programming languages such as C and C++.





1.12 Computer program:

- ❖ A program is a set of instructions following the rules of the chosen language.
- ❖ Without programs, computers are useless.
- ❖ A program is like a recipe. It contains a list of ingredients (called variables) and a list of directions (called statements) that tell the computer what to do with the variables.

1.13 Programming Language:

- ❖ A vocabulary and set of grammatical rules (syntax) for instructing a computer to perform specific tasks.
- ❖ Programming languages can be used to create computer programs.

- ❖ The term programming language usually refers to high-level languages, such as BASIC, C, C++, COBOL, FORTRAN, Ada, and Pascal.
- ❖ You eventually need to convert your program into machine language so that the computer can understand it.
- ❖ There are two ways to do this:
  - Compile the program
  - Interpret the program
- ❖ Compile is to transform a program written in a high-level programming language from source code into object code.
- ❖ This can be done by using a tool called compiler.
- ❖ A compiler reads the whole source code and translates it into a complete machine code program to perform the required tasks which is output as a new file.
- ❖ Interpreter is a program that executes instructions written in a high level language.
- ❖ An interpreter reads the source code one instruction or line at a time, converts this line into machine code and executes it.

1.14 Computer programming:

- ❖ Computer programming is the process of writing, testing, debugging/troubleshooting, and maintaining the source code of computer programs.
- ❖ This source code is written in a programming language like C++, JAVA, Perl etc.

1.15 Computer programmer:

- ❖ A programmer is someone who writes computer program.
- ❖ Computer programmers write, test and maintain programs or software that tells the computer what to do.

1.16 What skills are required to become a programmer:

☒ **Programming-** Writing computer programs for various purposes.

☒ **Writing-** Communicating effectively with others in writing as indicated by the needs of the audience.

☒ **Reading Comprehension** – Understanding written sentences and paragraphs in work-related documents.

☒ **Critical Thinking**- Using logic and analysis to identify the strengths and weaknesses of different approaches.

☒ **Computers and Electronics-** Knowledge of electric circuit boards, processors, chips, and computer hardware and software, including applications and programming.

☒ **Mathematics**- Knowledge of numbers, their operations, and interrelationships including arithmetic, algebra, geometry, calculus, statistics, and their applications.

☒ **Oral Expression**- The ability to communicate information and ideas in speaking so others will understand.

☒ **Oral Comprehension-** The ability to listen to and understand information and ideas presented through spoken words and sentences.

☒ **Written Expression**- The ability to communicate information and ideas in writing so others will understand.

☒ **Written Comprehension-** The ability to read and understand information and ideas presented in writing.

1.17 Number System

We are familiar with the decimal number system which is used in our day-to-day work. Ten digits are used to four decimal numbers. To represent these decimal digits, ten separate symbols 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9 are used. But a digital computer stores, understands and manipulates information composed of any zeros and ones. So, each decimal digit, letters, symbols etc. written by the programmer (an user) are converted to binary codes in the form of 0's and 1's within the computer. The no. system is divided into some categories according to the base (or radix) of the system as binary octal and hexadecimal. If a number system of base r is a system, then the system has r distinct symbols for r digits. The knowledge of the number system is essential to understand the operation of a computer.

Computer architecture supports following number systems.

1. Decimal number system (Base10)
2. Binary number system (Base 2)
3. Octal number system (Base8)
4. Hexadecimal (hex) number system (Base 16)

1. Decimal Number System

Decimal no. system have ten digits represented by 0,1,2,3,4,5,6,7,8 and 9. So, the base or radix of such system is 10.

In this system the successive position to the left of the decimal point represent units, tens, hundreds, thousands etc. For example, if we consider a decimal number 257, then the digit representations are

2	5	7
Hundred Position	Tens Position	Units Position

The weight of each digit of a number depends on its relative position within the number.

Example 1.1:

The weight of each digit of the decimal no. 6472

$$6472 = 6000+400+70+2$$

$$= 6X10^3+4X10^2+7X10^1+2X10^0$$

The weight of digits from right hand side are-

Weight of 1st digit = $2 * 10^0$

Weight of 2nd digit = $7 * 10^1$

Weight of 3rd digit = $4 * 10^2$

Weight of 4th digit = $6 * 10^3$

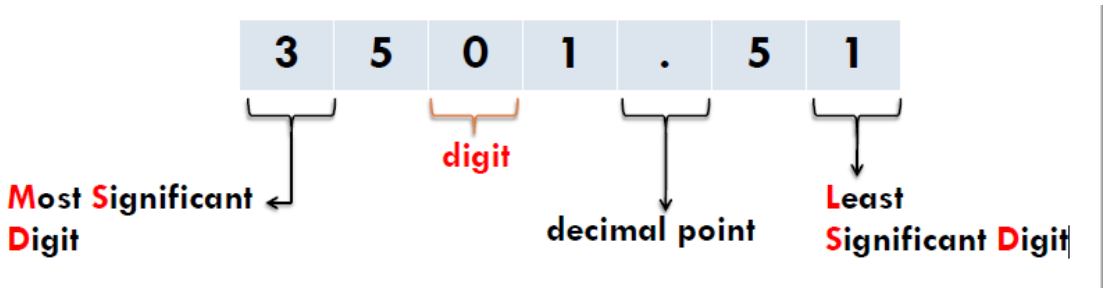
The above expressions can be written in general forms as the weight of n^{th} digit of the number from the right hand side

$= n^{th}$ digit $* 10^{n-1}$

$= n^{th}$ digit * (base) $^{n-1}$

The no. system in which the weight of each digit depends on its relative position within the number is called positional number system. The above form of general expression is true only for positional number system.

Example: 3501.51



2. *Binary Number System*

Only two digits 0 and 1 are used to represent a binary number system. So the base or radix of binary system is two (2). The digits 0 and 1 are called bits (Binary

Digits). In this number system the value of the digit will be two times greater than its predecessor. Thus the value of the places are:

<-- <-- 32 <-- 16 <--8 <--4 <--2 <--1

The weight of each binary bit depends on its relative position within the number. It is explained by the following example--

Example:

The weight of bits of the binary number 10110 is-

$= 1*2^4+0*2^3+1*2^2+1*2^1+0*2^0$

$= 16+0+4+2+0 = 22$(decimal number)

The weight of each bit of a binary no. depends on its relative pointer within the no. and explained from right hand side

Weight of 1st bit = 1st bit * 2^0

Weight of 2nd bit = 2nd bit * 2^1

and so on.

The weight of the nth bit of the number from right hand side

$=n^{th}$ bit X 2^{n-1}

$=n^{th}$ bit X $(Base)^{n-1}$

It is seen that this rule for a binary number is same as that for a decimal number system. The above rule holds good for any other positioned number system. The weight of a digit in any positioned number system depends on its relative position within the number and the base of the number system.

Example: 1101.01

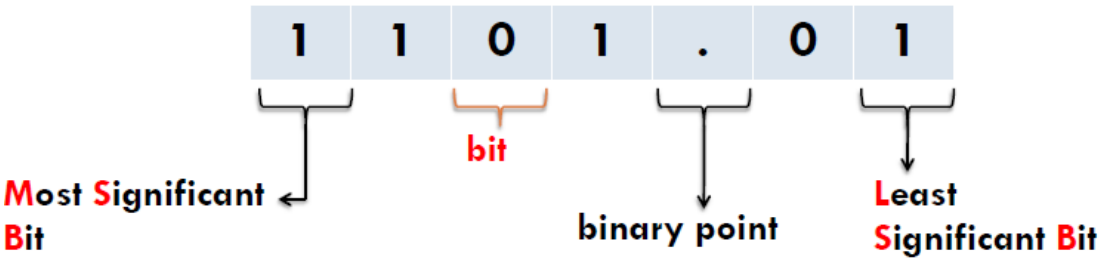


Table 1.1 shows the binary equivalent numbers for decimal digits.

Decimal Number	Equivalent Binary Number
0	*0*
1	*1*
2	*10*
3	*11*
4	*100*
5	*101*
6	*110*
7	*111*
8	*1000*
9	*1001*

3. ***Octal Number System***

A commonly used positional number system is the Octal Number System. This system has eight (8) digit representations as 0,1,2,3,4,5,6 and 7. The base or radix of this system is 8. The values increase from left to right as 1, 8, 64, 512, 4096 etc. The decimal value 8 is represented in octal as 10, 9 as 11, 10 as 12 and so on. As $8=2^3$, an octal number is represented by a group of three binary bits. For example 3 is represented as 011, 4 as 100 etc.

Table: The octal number and their binary representations.

Decimal Number.	Octal Number	Binary Coded Octal No.
0	*0*	*000*
1	*1*	*001*
2	*2*	*010*
3	*3*	*011*
4	*4*	*100*
5	*5*	*101*
6	*6*	*110*
7	*7*	*111*
8	*10*	*100 000*
10	*12*	*001 010*
15	*17*	*001 111*
16	*20*	*010 000*

4. Hexadecimal Number System

The hexadecimal number system is now extensively used in computer industry. Its base (or radix) is 16, ie. 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F. The hexadecimal numbers are used to represent binary numbers because of case of conversion and compactness. As $16=2^4$, hexadecimal number is represented by a group of four binary bits. For example, 5 is represented by 0101 .

Table 2.3 shows the binary equivalent of a decimal number and its hexadecimal representation .

Table: Hexadecimal number and their Binary representation

Decimal No.	*Hexadecimal No.*	*Binary coded Hex. No*
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Conclusions

☒ BINARY NUMBER SYSTEM

A Binary number system has only two digits that are **0 and 1**. Every number (value) represents with 0 and 1 in this number system. The base of binary number system is 2, because it has only two digits.

☒ OCTAL NUMBER SYSTEM

Octal number system has only eight (8) digits from **0 to 7**. Every number (value) represents with 0,1,2,3,4,5,6 and 7 in this number system. The base of octal number system is 8, because it has only 8 digits.

☒ DECIMAL NUMBER SYSTEM

Decimal number system has only ten (10) digits from **0 to 9**. Every number (value) represents with 0,1,2,3,4,5,6, 7,8 and 9 in this number system. The base of decimal number system is 10, because it has only 10 digits.

☒ HEXADECIMAL NUMBER SYSTEM

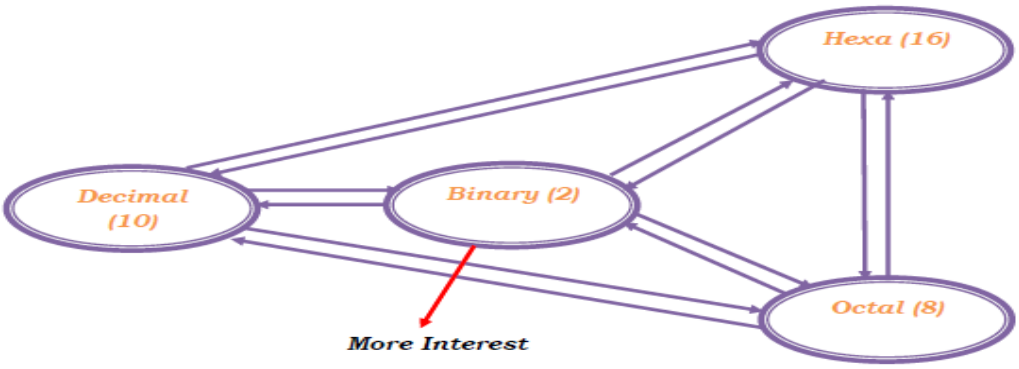
A Hexadecimal number system has sixteen (16) alphanumeric values from **0 to 9** and **A to F**. Every number (value) represents with 0,1,2,3,4,5,6, 7,8,9,A,B,C,D,E and F in this number system. The base of hexadecimal number system is 16, because it has 16 alphanumeric values. Here A is 10, B is 11, C is 12, D is 14, E is 15 and F is 16.

- **Decimal system** uses 10 symbols ***(digits)*** 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- **Octal System** uses eight symbols 0, 1, 2, 3, 4, 5, 6, 7
- **Binary System** uses only two symbols ***0 and 1***
- **Hexadecimal System** uses sixteen symbols

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

1.18 Number System Conversion

Convert a number from one number system to another



Why we need conversion?

✓ We need decimal system for ***real world*** (for presentation and input): for example: we use 10-based numbering system for input and output in digital calculator.

✓ We need binary system inside calculator for ***calculation***



1) *Binary to decimal conversions:*

✓ ***Rule***: any binary number can be converted to its decimal equivalent simply by ***summing together the weights of the various positions in the binary number which contains a 1***.

Example 1: Convert 11011_2 to its decimal equivalent

$$\begin{array}{ccccc} 1 & 1 & 0 & 1 & 1 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 2^4 & +2^3 & 0 & 2^1 & 2^0 \end{array} \quad = 16+8+2+1 = 27_{10}$$

Example 2: Convert 101101012 to decimal equivalent

$$2^7 + 0 + 2^5 + 2^4 + 0 + 2^2 + 0 + 2^0 = (181)_{10}$$

2) *Decimal to binary conversions:*

There are two ways to convert a decimal number to its equivalent binary representation

1. The reverse of the binary-to-decimal conversion process *(optional). The decimal number is simply expressed as a sum of powers of 2 and then* 1_2 *and* 0_2 *are written in the appropriate bit positions.*

Example 1:-Convert 45_{10} to binary number

$$45_{10} = 32 + 8 + 4 + 1 = 2^5 + 0 + 2^3 + 2^2 + 0 + 2^0 = 101101_{(2)}$$

Example 2:-Convert 76_{10} to binary number

$$76_{10} = 64 + 8 + 4 = 26 + 2^3 + 2^2 = 1001100_2$$

2. Repeated division*: Repeating division the decimal number by 2 and writing down the remainder after each division until a quotient of* **0** *is obtained.*

> **Note:**
> **The binary result is obtained by writing the first remainder as the LSB and the last remainder as the MSB.**

General Rule 1: Conversion from decimal to other base

1. Divide decimal number by the base (2, 8, 16,…).

2. The remainder is the lowest-order digit.

3. Repeat first two steps unit no divisor remains.

General Rule 2: Decimal fraction conversion to anther base

1. Multiply decimal number by the base (2, 8,…).

2. The integer is the highest-order digit.

3. Repeat first two steps until fraction becomes zero.

Example 1 Convert 25_{10} to binary number

$\frac{25}{2}$ = 12 + remainder of 1 (LSB)

$\frac{12}{2}$ = 6 + remainder of 0

$\frac{6}{2}$ = 3 + remainder of 0

$\frac{3}{2}$ = 1 + remainder of 1

$\frac{1}{2}$ = 0 + remainder of 1 (MSB)

$25_{10} = 1\ 1\ 0\ 0\ 1_2$

Example 2 Convert 13_{10} to binary number

Division by 2		Quotient integer	remainder
$\frac{13}{2}$	=	6	1 (a_0)
$\frac{6}{2}$		3	0 (a_1)
$\frac{3}{2}$		1	1 (a_2)
$\frac{1}{2}$		0	1 (a_3)
Answer		$(13)_{10} = (a_3\ a_2\ a_1\ a_0) = (1101)_2$	

Example 3: Convert 0.625_{10} to binary number

Multiply by 2	Integer		Fraction	coefficient
0.625*2 =	1	+	0.25	$a_1 = 1$
0.250*2 =	0	+	0.50	$a_2 = 0$
0.500*2 =	1	+	0(stop)	$a_3 = 1$
Answer	$(0.625)_{10} = (0.a_1\ a_2\ a_3)_2 = (0.101)_2$			

Correct order

3) *Octal-to-decimal*

✓ To convert, we need to ***multiply each octal digit by its positional weight***.

Example 1

$$372_{(8)} = (3*8^2) + (7*8^1) + (2*8^0) = (3*64) + 56+2 = 250_{10}$$

Example 2

$$24.6_{(8)} = (2*8^1) + (4*8^0) + (6*8^{-1}) = 20.75_{10}$$

4) *Decimal to octal*

✓ ***Repeated division by 8.***

Example 1: Convert 266_{10} to octal number.

$\frac{266}{8} = 33$ + remainder of 2 (LSD)

$\frac{33}{8} = 4$ + remainder of 1

$\frac{4}{8} = 0$ + remainder of 4

$266_{10} = 4\ 1\ 2_{(8)}$

Example 2: Convert 0.35_{10} to octal number.

Multiply by 8	Integer		Fraction	coefficient
0.35*8=	2	+	0.80	$a_1 = 2$
0.8*8 =	6	+	0.40	$a_2 = 6$
0.4*8 =	3	+	0.20	$a_3 = 3$
0.2*8 =	1	+	0.60	$a_4 = 1$
0.6*8 =	4	+	0.80	$a_5 = 4$
Answer	$(0.35)_{10} = (0.a_1\ a_2\ a_3\ a_4\ a_5)_2 = (0.26314)_8$			

Repeated "stop"

Correct order

5) ***Hexa-to-decimal***

Example 1:-Convert $356_{(16)}$ to decimal:

$$356_{(16)} = (3*16^2) + (5*16^1) + (6*16^0) = 3*256 + 80 + 6 = 854_{(10)}$$

Example 2:-Convert $2AF_{(16)}$ to decimal:

$$2AF_{(16)} =$$

$$(2*16^2) + (10*16^1) + (15*16^0) = 512+160+15 = 687_{(10)}$$

6) ***Decimal-to-hexa:(using repeated division by 16)***

Example 1: Convert 423_{10} to hex number.

$\frac{423}{16}$ = 26 + remainder of 7 (LSD)

$\frac{26}{16}$ = 1 + remainder of 10

$\frac{1}{16}$ = 0 + remainder of 1

$423_{10} = 1\ A\ 7_{(16)}$

7) ***Hexa-to-binary:***

✓ Each hexa digit is converted to its ***four-bit binary equivalent***:

Example 1: Convert $9F2_{(16)}$ to its binary equivalent

9 F 2= 1001 1111 0010

$9F2_{(16)} = 100111110010_{(2)}$

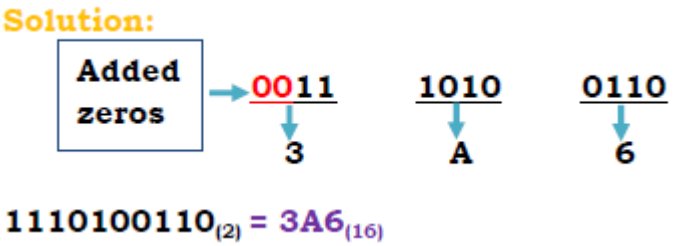
Example 2: Convert $BA6_{(16)}$ to binary equivalent

$BA6_{(16)} = (1011\ 1010\ 0110)_2$

8) *Binary-to-hexa*

- ***The binary numbers are grouped into groups of four bits*** and each group is converted to its equivalent hexa digit.
- ***Zeros*** are added as needed to complete a four-bit group.

Example 1: Convert $1110100110_{(2)}$ to hexa equivalent



$1110100110_{(2)} = 3A6_{(16)}$

Example 2: Convert 101011111_2 to hexa equivalent

Solution:

$1\ 0101\ 1111_2 = 15F_{(16)}$

9) *Octal to binary conversion:*

✓ Conversion each octal digit to its ***three bit binary equivalent***

Conversion Table								
Octal digit	0	1	2	3	4	5	6	7
Binary equivalent	000	001	010	011	100	101	110	111

Using this table, we can convert any octal number to binary by individually converting each digit.

Example 1: **Convert $472_{(8)}$ to binary number**

Solution:

4	7	2
100	111	010

.: $472_{(8)} = 100111010_{(2)}$

Example 2: **Convert $5431_{(8)}$ to binary number**

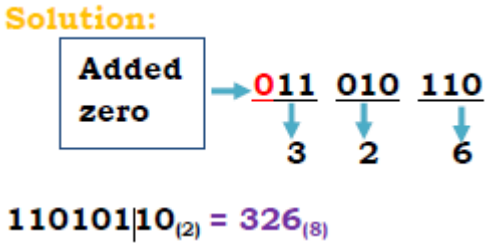
Solution:

$5431_{(8)} = 101\ 100\ 011\ 001 = 101100011001_{(2)}$

10) *Binary to octal conversion:*

The bits of the binary number are grouped into group of 3 bits starting at the LSB, then each group is converted to its octal equivalent (see table).

Example 1: Convert $11010110_{(2)}$ to octal equivalent



Note:

Zero was placed to the left of the MSB to produce groups of 3 bits.

General example:

Convert 177_{10} to its eight-bit binary equivalent by first converting to octal.

Solution:

$\frac{177}{8}$ = 22 + remainder of 1 (LSD)

$\frac{22}{8}$ = 2 + remainder of 6

$\frac{2}{8}$ = 0 + remainder of 2

$177_{10} = 2\ 6\ 1_{(8)}$

✓ Thus $177_{10} = 261_{(8)}$, now we can quickly convert this octal number to its binary equivalent **010110001** to get eight bit representation. So:

$177_{10} = 1011000_{(2)}$

Important Note: *this method of decimal-to-octal-to-binary conversion is often quicker than going directly from decimal to binary, especially for large numbers.*

General questions of chapter 1

1. Complete the following sentences:

1) A …… is an electronic device, operating under the control of instructions stored in its own memory.

2) Any digital computer carries out functions likes that…… ..and …… and …….

3) Any kind of computers consists of …… and …….

4) ……. is the collection of physical elements that constitutes a computer system.

5) …… is any peripheral (piece of computer hardware equipment to provide data and control signals to an information processing system such as a computer or other information appliance.

6) A …… is brain of a computer. It is responsible for all functions and processes. Regarding computing power, the CPU is the most important element of a computer system.

7) ..... Executes all arithmetic and logical operations.

8) ..... Controls and co-ordinates computer components.

9) …… Stores the data that is to be executed next, "very fast storage area".

10) …… is volatile in nature, which means that data will be erased once supply to the storage device is turned off.

11) ..... is a permanent form of storage.

12) …… is any piece of computer hardware equipment used to communicate the results of data processing carried out by an information processing system.

13) ……is a generic term for organized collections of computer data and instructions.

14) Software is divided into …… and …….

15) ..... A small, single-user computer based on a microprocessor.

16) ……A powerful multi-user computer capable of supporting many hundreds or thousands of users simultaneously.

17) ……An extremely fast computer that can perform hundreds of millions of instructions per second.

18) The basic unit used in computer data storage is called …….

19) The speed of Central Processing Unit (CPU) is measured by …….

20) …….or machine code is the native language directly understood by the computer's Central Processing Unit or CPU.

21) …… is a set of codes that can run directly on the computer's processor.

22) …… are user-friendly languages which are similar to English with vocabulary of words and symbols.

23) …… translates high level language programs directly into machine language program. This process is called …….

24) …….translates high-level instructions into an intermediate form, which it then executes.

25) A …… is a set of instructions following the rules of the chosen language.

26) The term programming language usually refers to high-level languages, such as……, and …….

27) A **.....**……is someone who writes computer program.

28) There are three types of programming language: ……, Assembly language and ……

29) ……is a collection of binary digits or bits that the computer reads and interprets.

30) Assembly language programs are translated into machine language

by a program called …….

31) …… are numeric, alphabetic data and …….

32) .....are used for internal representation of data in computers.

33) ..... can represent 64 different characters.

34) ASCII is of two types …… and …….

35) First.....in ASCII-7 and ASCII-8 are the same.

36) $(47692)_{10}$ is (……..)$_{10}$

37) $(10111001)_2$ is (……)$_{10}$

38) $(12403)_8$ is (……)$_{10}$

39) $(FB40A)_{16}$ is (……)$_{10}$

40) $(1020)_{10}$ is (… . ……..)$_8$

2. Draw a figure for the following terms:

1) CPU.

2) Starting from hardware and reaching high level language.

3) Compilation process.

4) Interpretation process.

3. Compare between the following:

1) Machine and assembly languages. From where (time to execute, time to develop and translator)

2) Machine and high level languages. From where (time to execute, time to develop and translator)

3) Primary and secondary memories. From where (examples)

4) RAM and ROM. From where (volatile and read and write memories)

5) System and application software. From where (examples)

4. Rearrange the following units (Ascending order):

1) Computer storage units.

2) CPU speed measures.

3) Computer languages.

5. State the Program translators (Language Processors) and its types?

6. Define Assembler ,draw block diagram to show the process؟

7. What skills are required to become a programmer؟

8. Define : Computer programming؟

9. Define :Programming Language:

10. What is the Functionalities of a computer?

11. What is the Characteristics of computers?

12. What is the computer classification?

13. What are the Application of computers?

Sheet Number Systems – Conversion & Math Practice Problems

Conversion Problems

1. Convert each of the following binary numbers to octal, decimal, and hexadecimal formats.

1. $(111011101)_2$
2. $(10101010111)_2$
3. $(111100000)_2$

2. Convert each of the following octal numbers to binary, decimal, and hexadecimal formats.

1. $(3754)_8$
2. $(7777)_8$
3. $(247)_8$

3. Convert each of the following decimal numbers to binary, octal, and hexadecimal formats.

1. $(3479)_{10}$
2. $(642)_{10}$
3. $(555)_{10}$

4. Convert each of the following hexadecimal numbers to binary, octal, and decimal formats.

1. $(4FB2)_{16}$
2. $(88BAE)_{16}$
3. $(DC4)_{16}$

CHAPTER (2)

Structured Function

CHAPTER 2 : Structured programs

2.1 Structured programs:

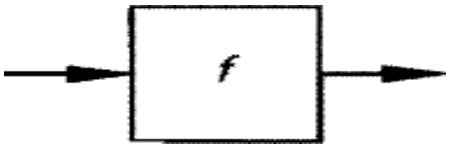
Structure theorem which shows that the effect on data of any arbitrarily complex control structure can be accomplished by the use of **process design language (PDL)** control structures alone. The Structure Theorem proof construction permits systematic transformation of programs expressed in these arbitrarily complex control structures into equivalent programs expressed in the PDL control structures. In order to make these ideas precise, it introduces three fundamental concepts: **proper programs**, **prime programs** and **program functions**.

2.2 Programs execution:

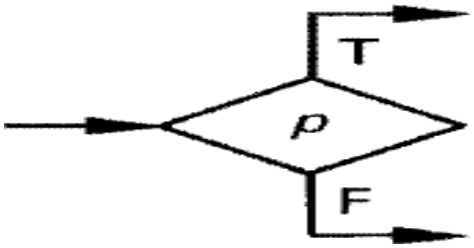
2.2.1 Flowchart programs:

A flowchart is a directed graph that depicts the flow of execution control of program and the instructions to be executed. Each instruction of a program corresponds to **a node** in the flowchart; each possible flow of control corresponds to a line. If an instruction node has more than one out-line, it is control instruction. If the execution of a control instruction affects no data except an instruction counter, it is a **pure control instruction**; otherwise control instruction has **side effects** that change data values. (Note that if real-time clock is part of the data, then no pure control instructions exist). Isolation of side effects permits better human understanding of programs. Side effects can be useful in control instructions-for example, in looping (decrement and branch), in subroutine linkages (branch and link return), in implementing push down stacks-but we do not pursue those ideas further here.

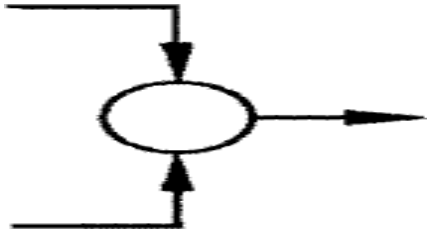
❖ If a flowchart node has a single in-line and single out-line, it is called **a function node**. Where a function named f is associated with the node, typically an assignment instruction. The term function node is especially appropriate since any assignment instruction can be represented entirely by a mathematical function in its effect on data.



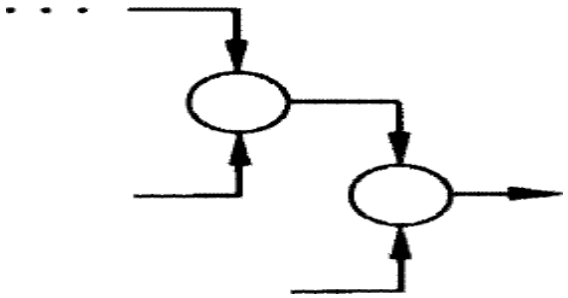
❖ If a flowchart node has a single in-line and two out-lines and is a pure control instruction, it is called **a predicate node**. Where a predicate named p is associated with the node. A predicate node ' directs the flow of execution control according to whether the predicate evaluates to true or false (T or F); it does not affect the data of a program otherwise. The true out-line will always be drawn above the false out-line.



❖ It is convenient to introduce one more node with a "no-op" instruction that does not change or evaluate data, with two in-lines and one out-line, called **a collecting node**:



❖ In fact, a flowchart may join more than two in-lines at an instruction, but any number of in-lines can be depicted by a structure of multiple collecting nodes:



❖ The **control structure** of a flowchart preserves the ordering of function nodes, predicate nodes, and collecting nodes, but ignores the identity of associated functions, predicates, and predicate values. The flowchart illustrated in figure below, with predicates named p and q and functions named g and h,

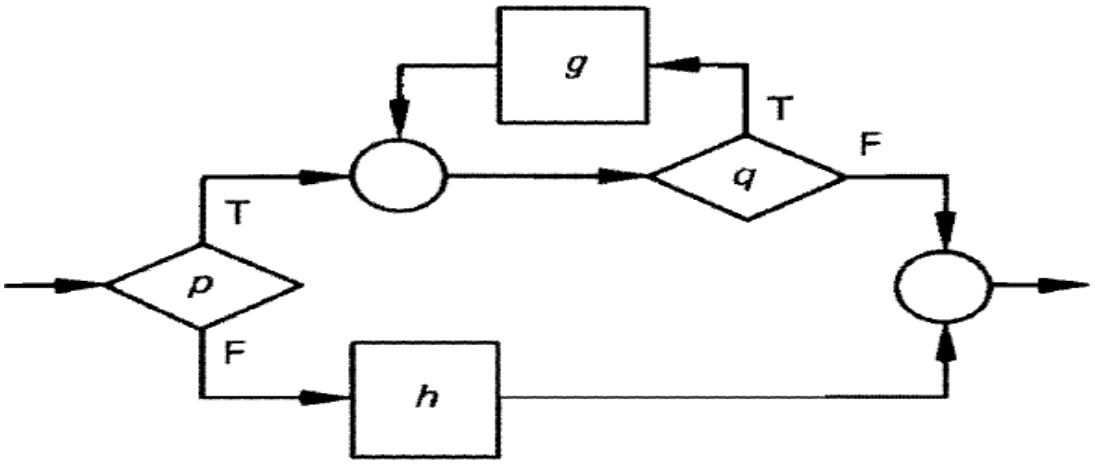


Fig. 1: Flowchart with control structure.

❖ The above flowchart has a control structure depicted by, say, either of the diagrams in figure below. We note in passing that a control structure has **a natural dual** obtained by reversing each line and interchanging the roles of predicate and collecting nodes.

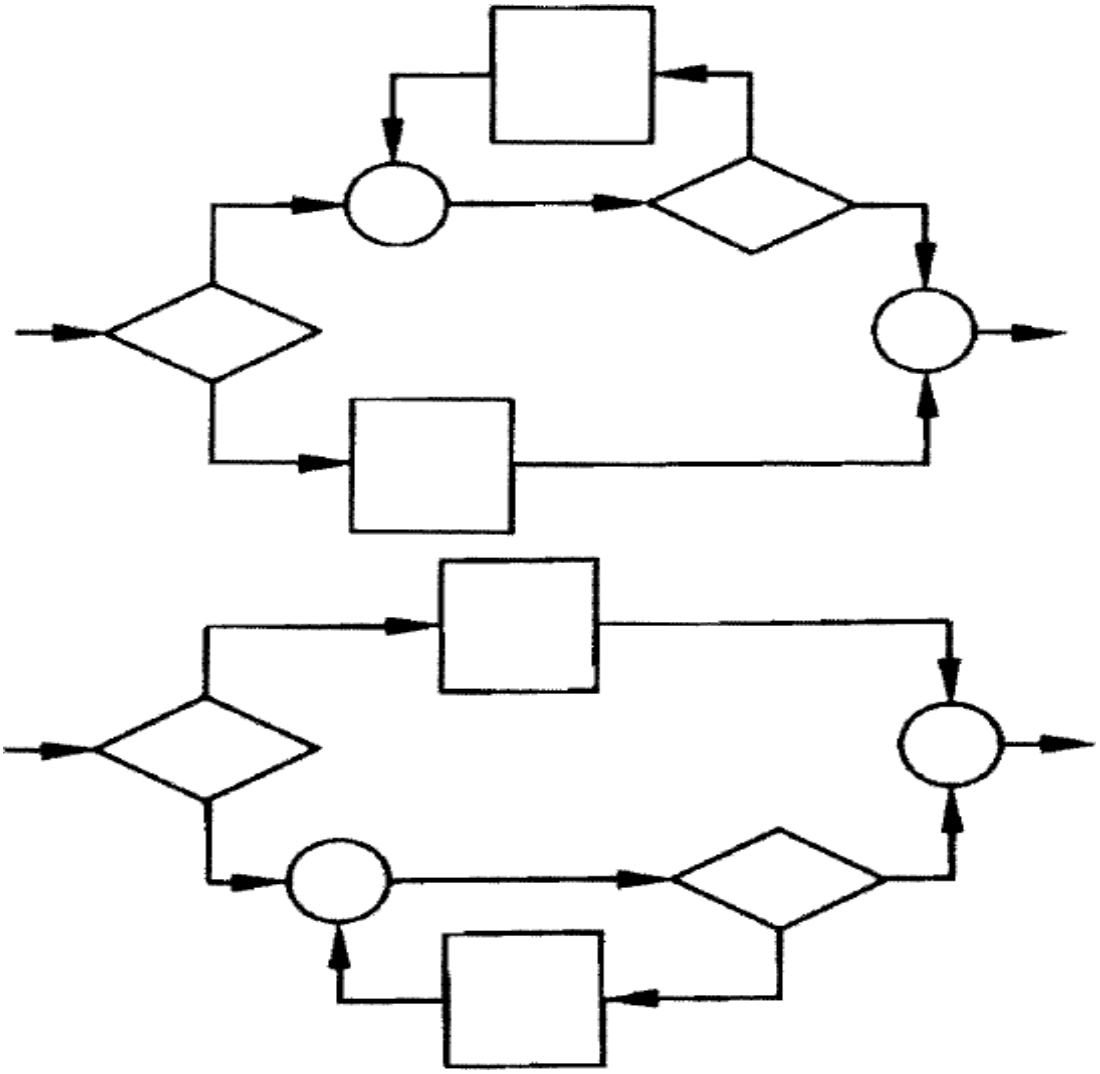


Fig. 2: Control structure with natural dual.

2.2.2 Proper programs:

A **proper program** is a program with a control structure that:

A. has a single entry line and a single exit line, and

B. for each node, has a path through that node from the entry line to the exit line.

Condition 2 outlaws control structures such as those shown in figure below that have unleavable node sets (left) and unreachable node sets (right).

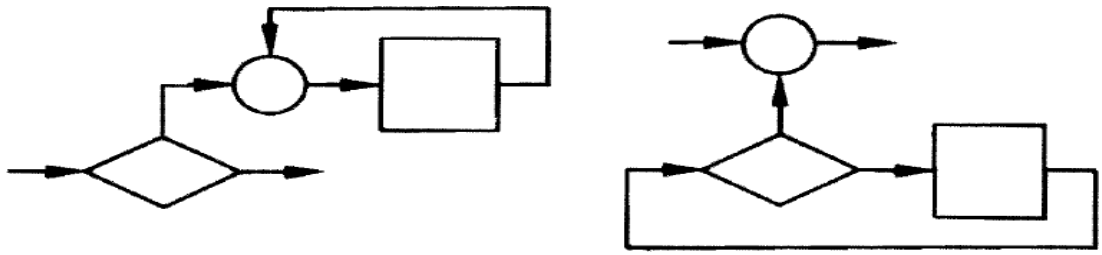


Fig. 3: Outlaws control structure.

A proper program shown below can be abstracted into a single -function node for better human understanding. The function node summarizes the total effect of the data operations and tests of the proper program it represents. For example, the proper program .

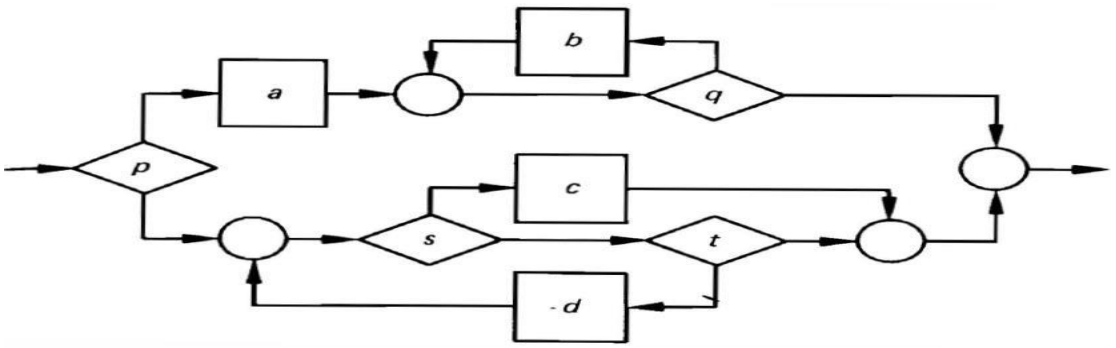
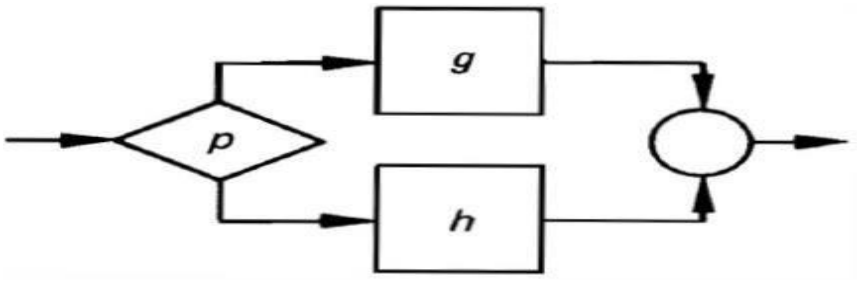
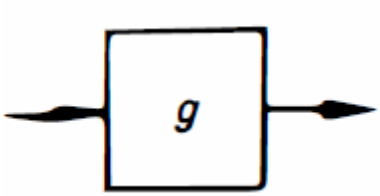


Fig. 4: Proper program.

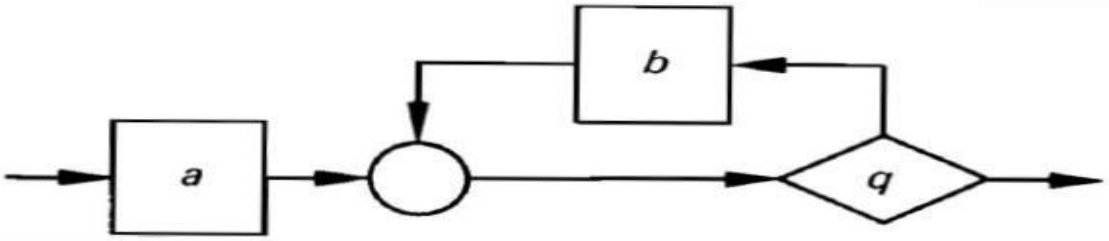
can be redescribed as



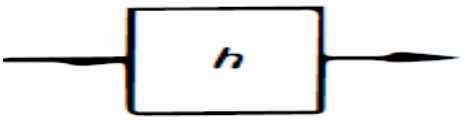
Where



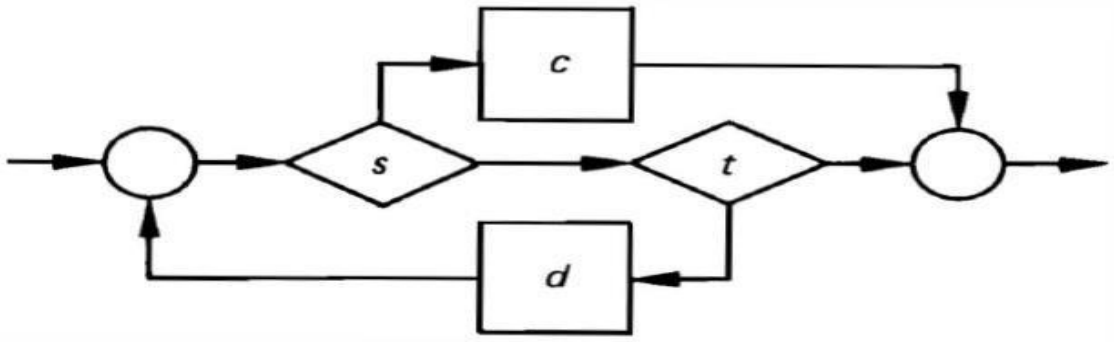
abstracts the proper program



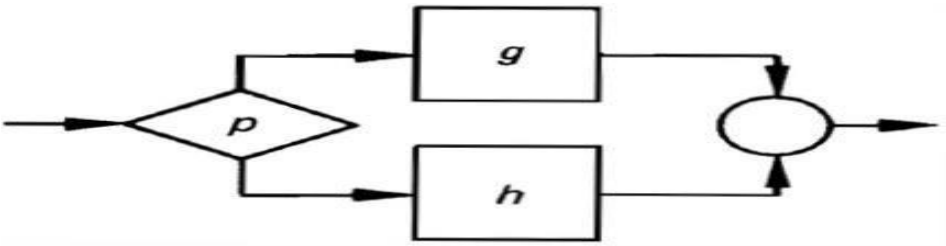
And



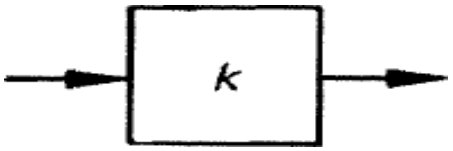
abstracts the proper program



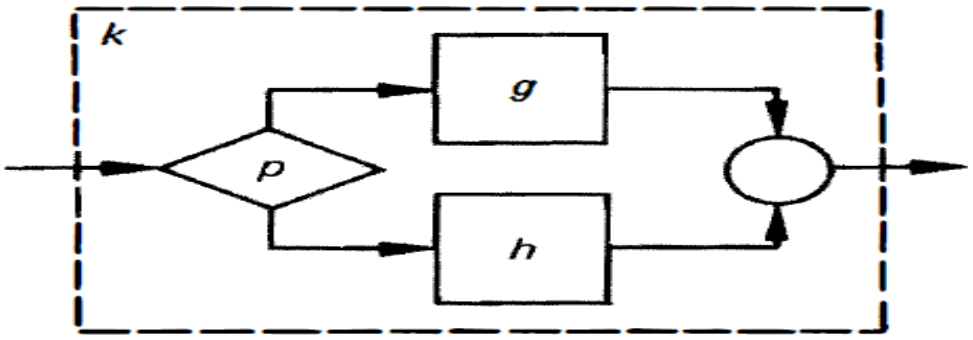
The reduced proper program



can then be abstracted to a single function node:



Conversely, any function node of a program can be expanded into a proper program without affecting the function of other parts of the program. For example, the function node just above could be expanded to, say



The parts of a program that are themselves proper are called proper subprograms. The proper subprograms of program k are g, h, a, b, c, and d. Predicates p, q, s, and t are not proper subprograms by themselves because they each have two out-lines.

C. Execution charts and trees:

A flowchart program defines execution sequences along paths and cycles. These execution sequences can often be better understood in terms of a finite tree called an **execution chart (E-chart)**. Given a proper flowchart, we construct its E-chart of nodes and lines in a stepwise manner as follows:

1) Start the E-chart with the entry line of the flowchart and its adjacent predicate, function, or collecting node.
2) At each step of construction, consider each execution path (a

directed path of lines and nodes adjoined to the entry line) in the evolving E-chart. If an execution path currently terminates in a function, predicate, or collecting node not found earlier in that specific path, adjoin all of that node's out-lines (from the flowchart) and the nodes with which those out-lines connect, if any, to the execution path.

3) When all execution paths terminate in exit lines or in nodes that previously appeared on the path, the E-chart is complete.

It is clear that this procedure terminates with a finite tree since each path will eventually exhaust the nodes of the flowchart. For example, for the flowchart illustrated in the first figure below (with numbered collecting nodes) the above procedure generates the E-chart shown in the second figure below.

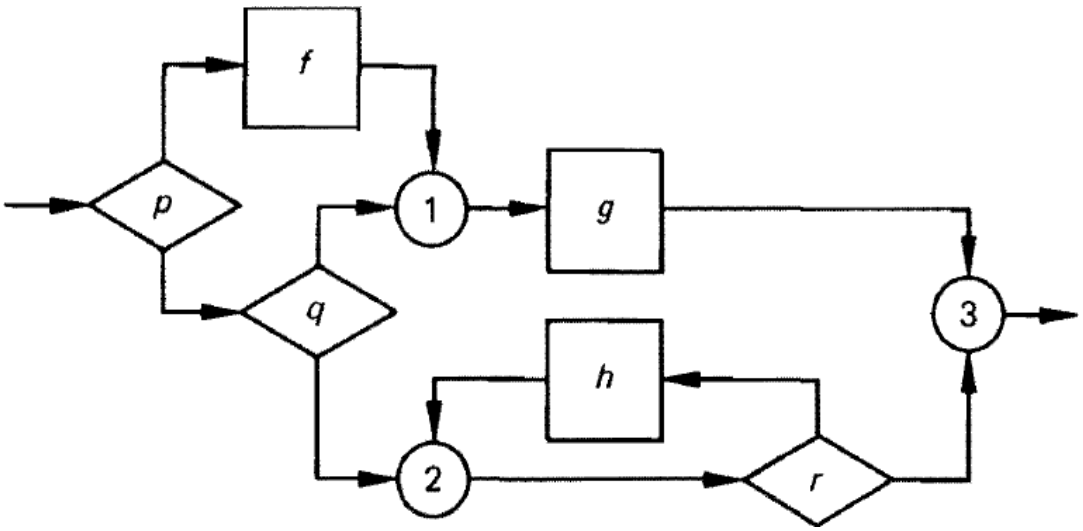
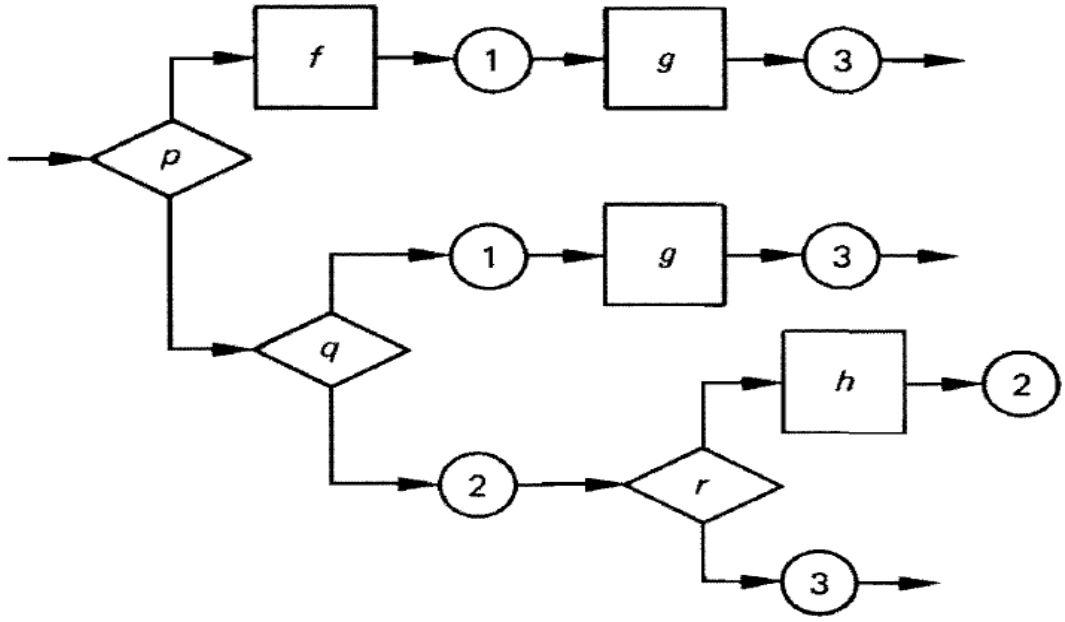


Fig. 5: A flowchart program.

The E-chart is:



For example, the E-chart in figure above can be simplified as illustrated in figure below, where only collecting node 2 has been retained.

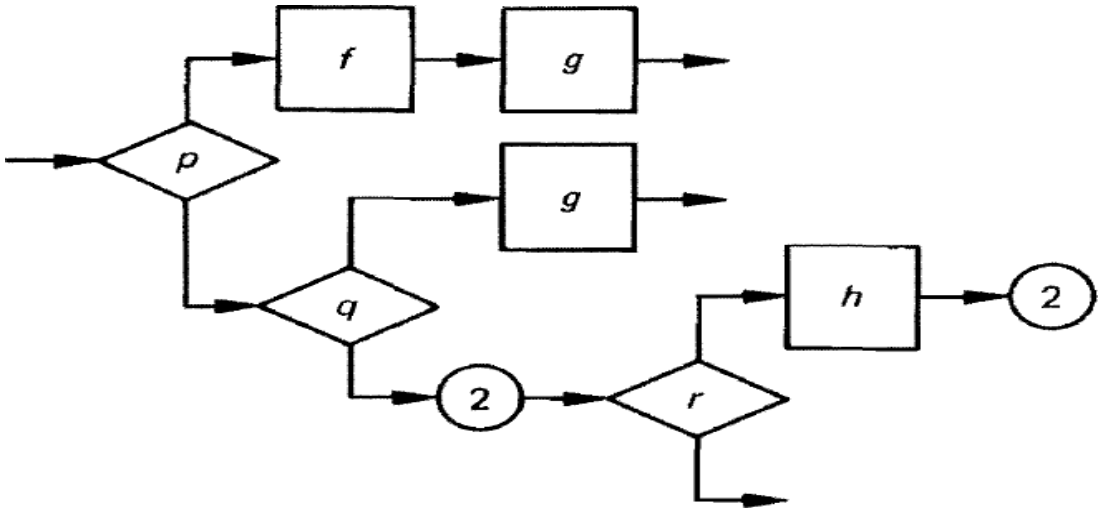


Fig.6: Simplified E-chart.

An **execution tree (E-tree)** of a flowchart program is a tree whose paths depict all possible execution sequences of the flowchart without retracing. If **loops are not present** in the flowchart, then the corresponding finite E-chart is the E-tree. If **loops are present** in a

flowchart, the E-tree is the infinite tree obtained by repeatedly replacing each repeated node by the subtree beginning at the first occurrence of that node earlier in the path. As a final simplification, the first occurrence of each repeated node can itself be suppressed. For example, the flowchart:

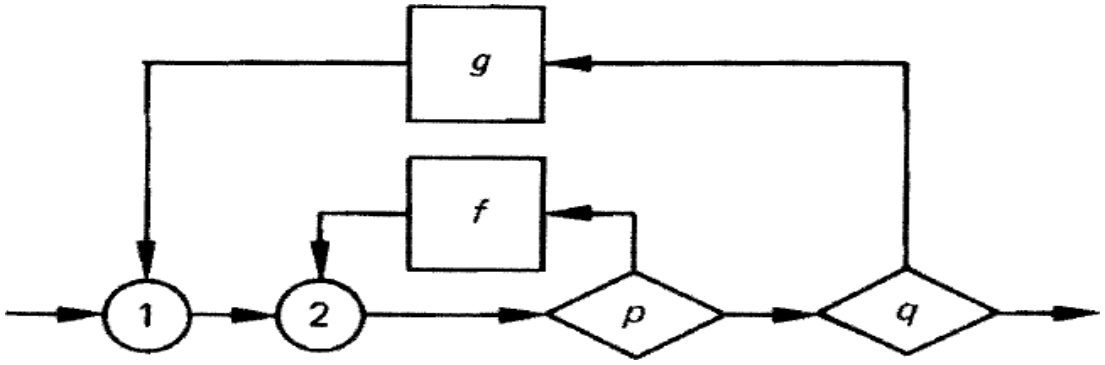


Fig. 7: A flowchart program.

Has the E-chart shown below:

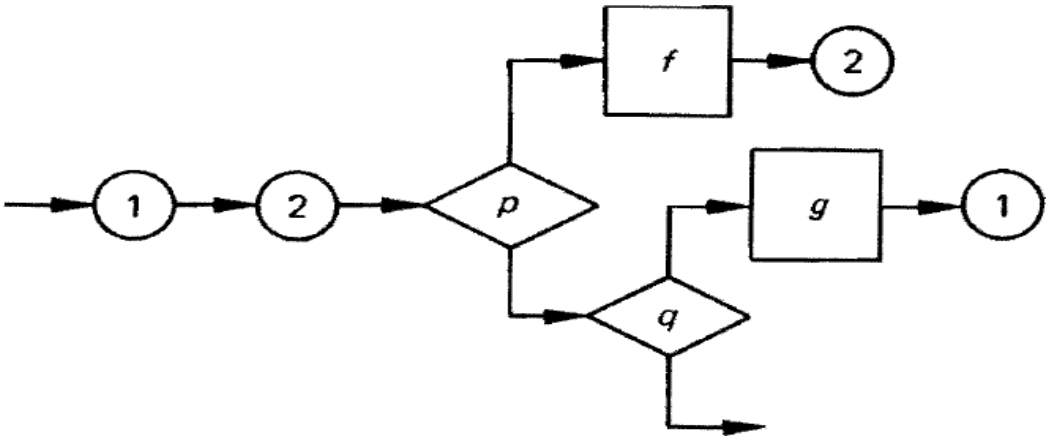


Fig. 8: The E-chart.

For example, consider the flowchart program shown in figure below. The E-chart, with end points labeled (1) to (5),

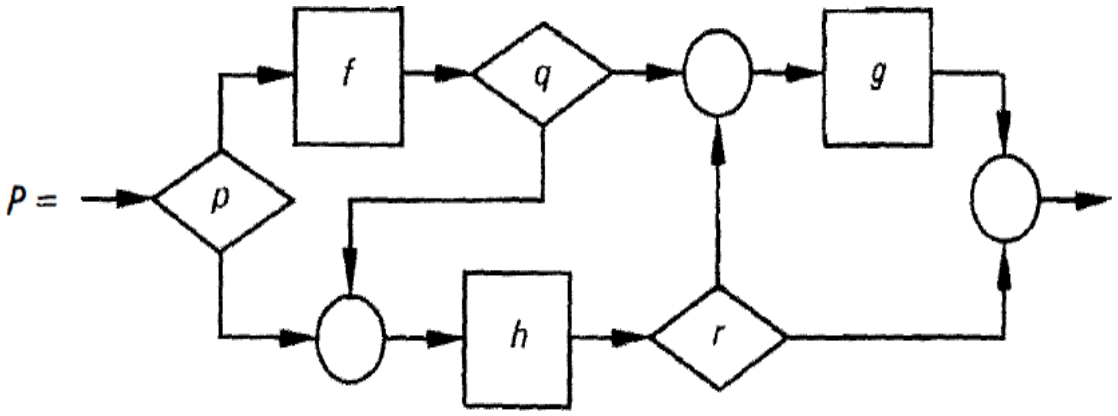


Fig. 9: A flowchart program.

The E-chart is illustrated below:

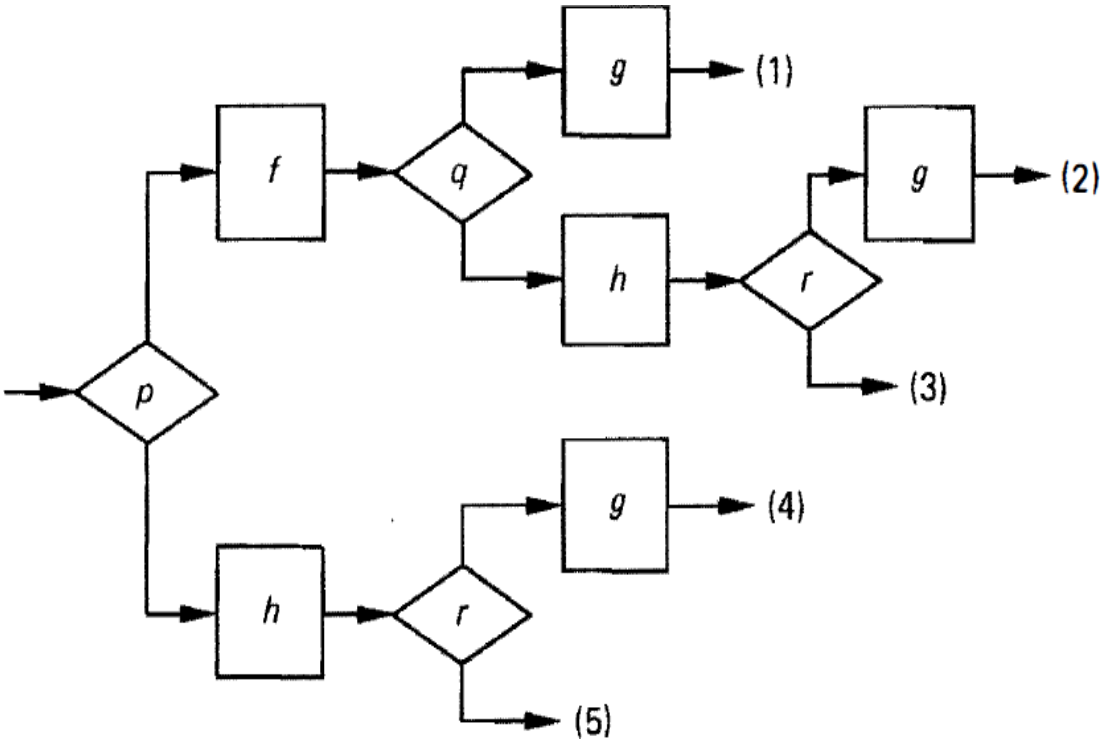


Fig. 10: The E-chart.

2.3 Program functions:

A- Relations:

To illustrate, consider the relations defined by:

equal $\{(x, y) | x = y\}$
less than $\{(x, y) | y < x\}$
opposite sign $\{(x, y) | x * y < 0\}$

We find that

equal is a reflexive relation $(7 = 7$ is *true*$)$
less than is not a reflexive relation $(7 < 7$ is *false*$)$
equal is a symmetric relation $((2/4 = 1/2) \rightarrow (1/2 = 2/4))$
less than is not a symmetric relation (it is *false* that $(2 < 7) \rightarrow (7 < 2)$)
less than is a transitive relation $(((2 < x) \wedge (x < y)) \rightarrow (2 < y))$

B- Functions:

A function is a relation, say f, such that for each $X \in D(f)$ there exists a unique element(X,Y) € F. We often express this as $= F(X)$, where Y is the unique value corresponding to X in the function. In illustration, the function

$$f = \{(x, y) | x \in \{0, 1\}, y = x^2 + 3x + 2\}$$

where $(x \in \{0, 1\})$ denotes the domain and $(y = x^2 + 3x + 2)$ denotes the rule, can also be given by enumeration as

$$f = \{(0, 2), (1, 6)\}$$

or by a set description with rule denoted by two equations:

$$f = \{(x, y) | x(x - 1) = 0, y - x^2 - 3x - 2 = 0\}$$

C- State machines:

A state machine is a function whose members are ordered pairs of ordered pairs, say

$$m = \{((x, y), (u, v))\}$$

That is also called a transition function. These members are customarily: given an interpretation denoted by

$$m = \{((\text{state, input}), (\text{newstate, output}))\}$$

State machines are useful in program design

D- List structures:

- The empty list, denoted by φ, is a sequence of no items.
- The fundamental relationship in lists is between members of the alphabet and a list, namely, being the first item, say a, of a nonempty list, say L, written

$$a = \text{head}(L),\ L \neq \emptyset$$

- A nonempty list L with its first member removed is written tail (L). Note that tail (L) can be the empty list. An item of a list, say a, is different from a one- element list, say (a), holding that same item; head (L) is an item, while tail (L) is a list. If L = φ , then head (L) is undefined.

E- Regular expressions:

A regular expression is the right-hand side of a single production which defines a language with operations of concatenation (language product), union, and iteration. A regular expression is a

compact way of describing a language.

There is an interesting **application of regular expressions** to program control structures. The instructions executed in a program form a string, and the strings formed over all possible executions form a language. The regular expressions and program control structures shown in figure below.

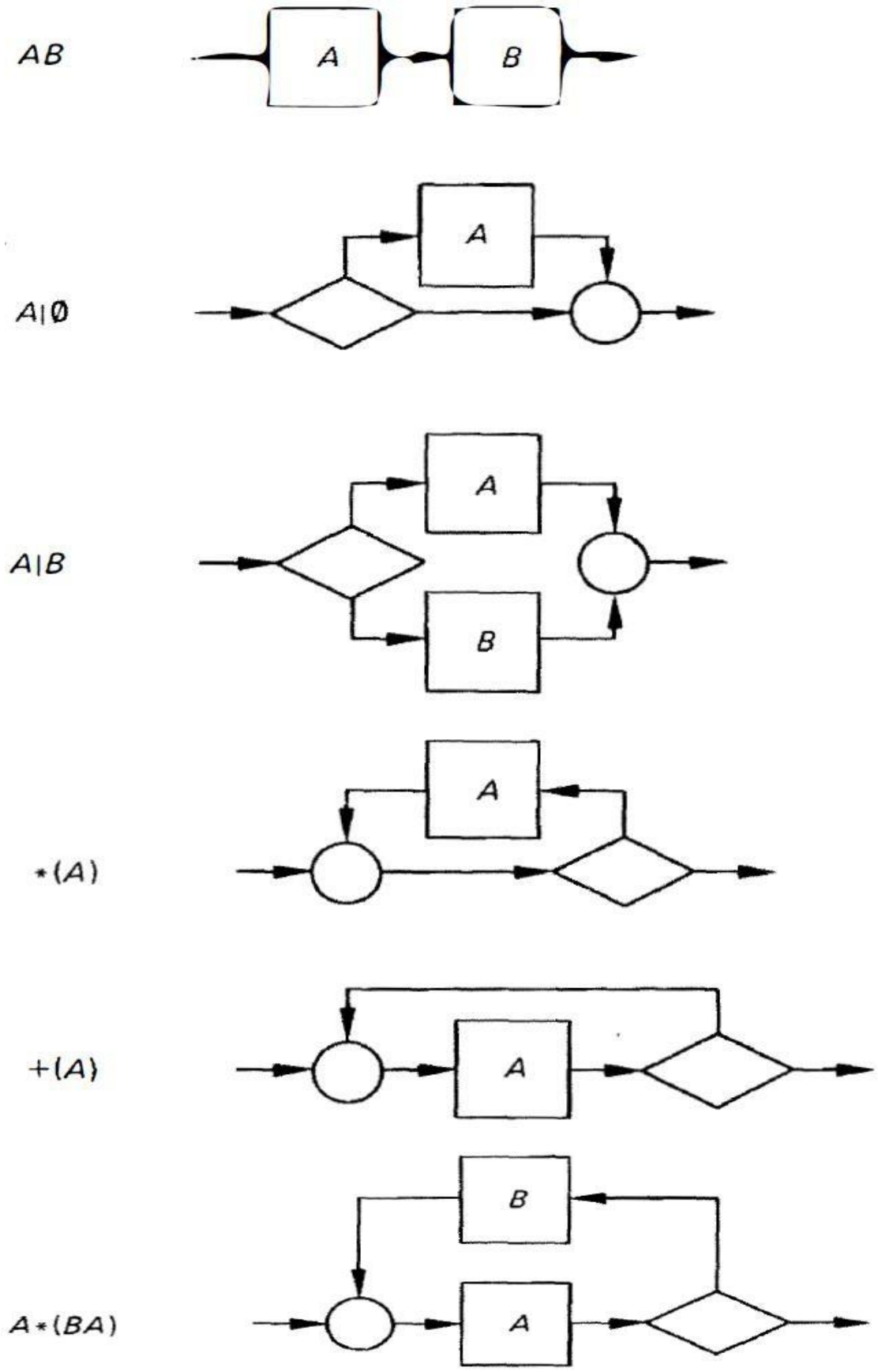


Fig. 11: The regular expressions and program control structures.

Regular expressions for complex program logic composed of these or similar structures can often be simplified, thus producing simpler program logic.

For example, the program in figure below has regular expression (ignoring predicate values)

$$((BA)|(CA))((A*(A))|\emptyset)$$

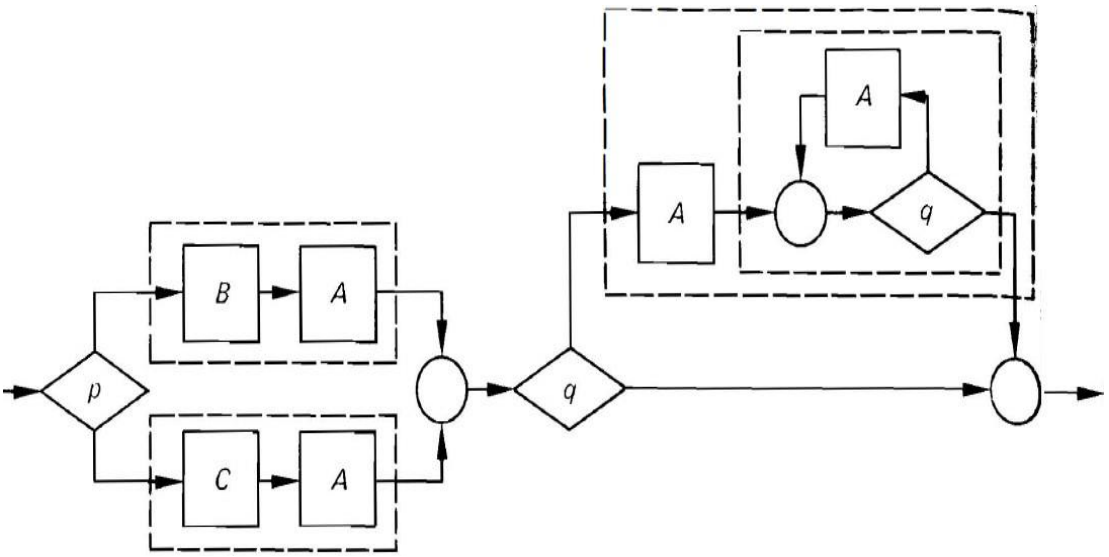


Fig. 12: A program with regular expression.

That can be simplified to as shown in equations and figure below, respectively:

$$((BA)|(CA))(+(A)|\emptyset)$$
$$= ((BA)|(CA))\,*(A)$$
$$= ((B|C)A)\,*(A)$$
$$= (B|C)(A\,*(A))$$
$$= (B|C)+(A)$$

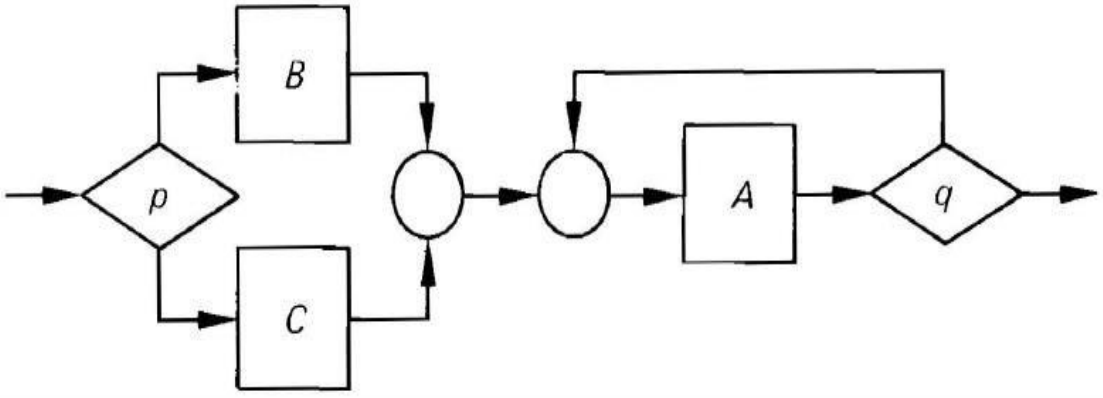


Fig. 13: The simplification for the program with regular expression.

F- Program effects on data:

An execution of a program may terminate (reach an end point) in its execution tree, and possibly every execution will terminate, even though the execution tree is itself infinite. Consider all possible executions defined by an execution tree. For each initial data state X for which execution terminates, a final data state Y is determined. The value Y is unique, given X, so that the set of all ordered pairs {(X, Y)} so defined is a function. We call this function the **program function** of a program.

- Given a proper program named P, we denote its program function by [P] (read "bracket P"). By definition then, a program

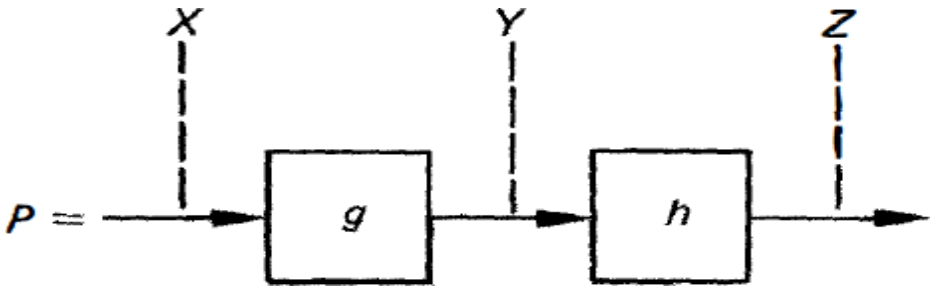
P composed of an individual function node f = {(X, Y)}



has program function:

$$[P] = \{(X, Y) \mid Y = f(X)\}$$

❖ A sequence of two functions:



Where g = {(X, Y)}, h = {(Y, Z)}, has as its program function the composition of the individual functions:

$$[P] = \{(X, Z) \mid Z = h(g(X))\}$$

For example, consider the flowchart program shown in figure below.

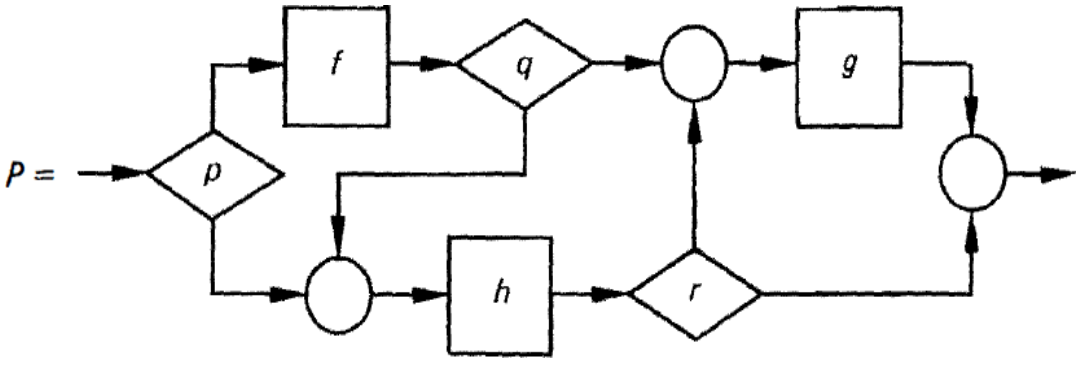


Fig. 14: A flowchart program.

The E-chart, with end points labeled (1) to (5), is shown in figure below:

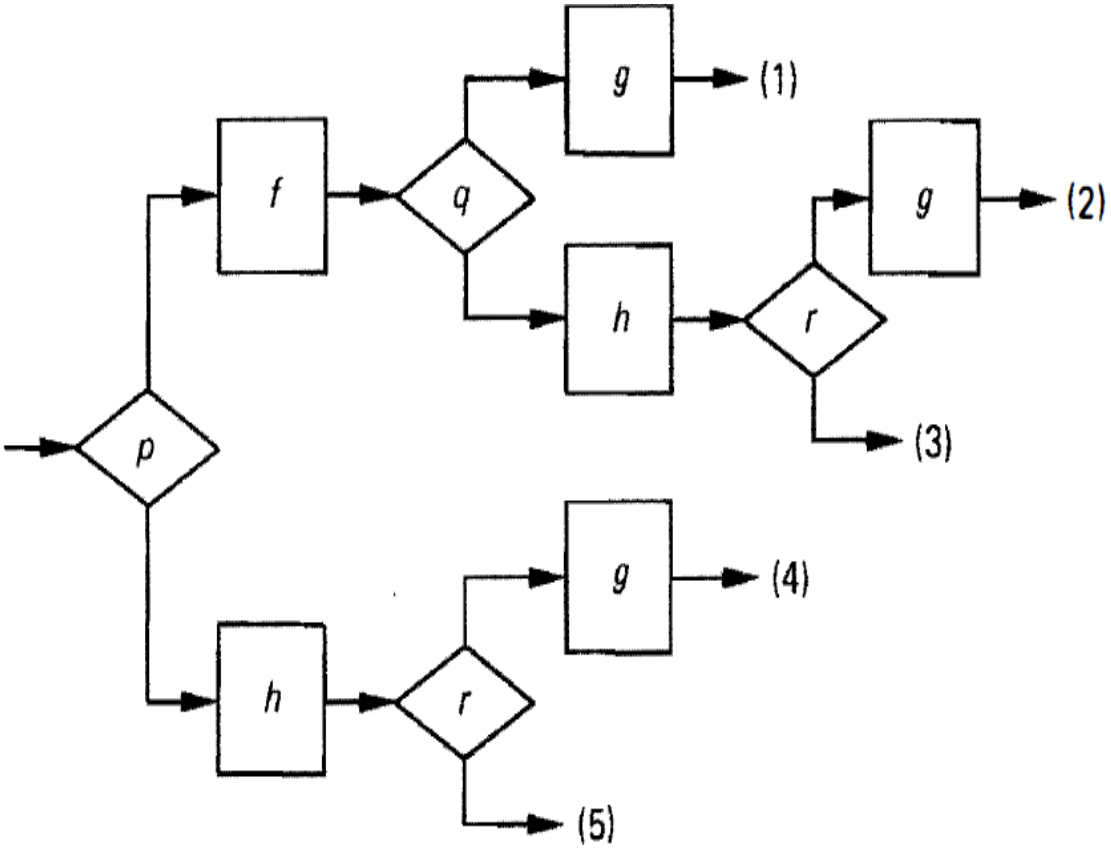


Fig. 15: The E-chart.

The program function is thus a union of five function compositions, each of whose domain and values are defined by a path through the E-chart; namely,

$$(1)\ \{(X, Y) \mid p(X) \wedge q(f(X)) \wedge Y = g(f(X))\}$$

$$(2)\ \{(X, Y) \mid p(X) \wedge \sim q(f(X)) \wedge r(h(f(X))) \wedge Y = g(h(f(X)))\}$$

$$(3)\ \{(X, Y) \mid p(X) \wedge \sim q(f(X)) \wedge \sim r(h(f(X))) \wedge Y = h(f(X))\}$$

$$(4)\ \{(X, Y) \mid \sim p(X) \wedge r(h(X)) \wedge Y = g(h(X))\}$$

$$(5)\ \{(X, Y) \mid \sim p(X) \wedge \sim r(h(X)) \wedge Y = h(X)\}$$

2.4 Prime programs:

A prime program is a proper program that has no proper subprogram of more than one node as illustrated in figure below:

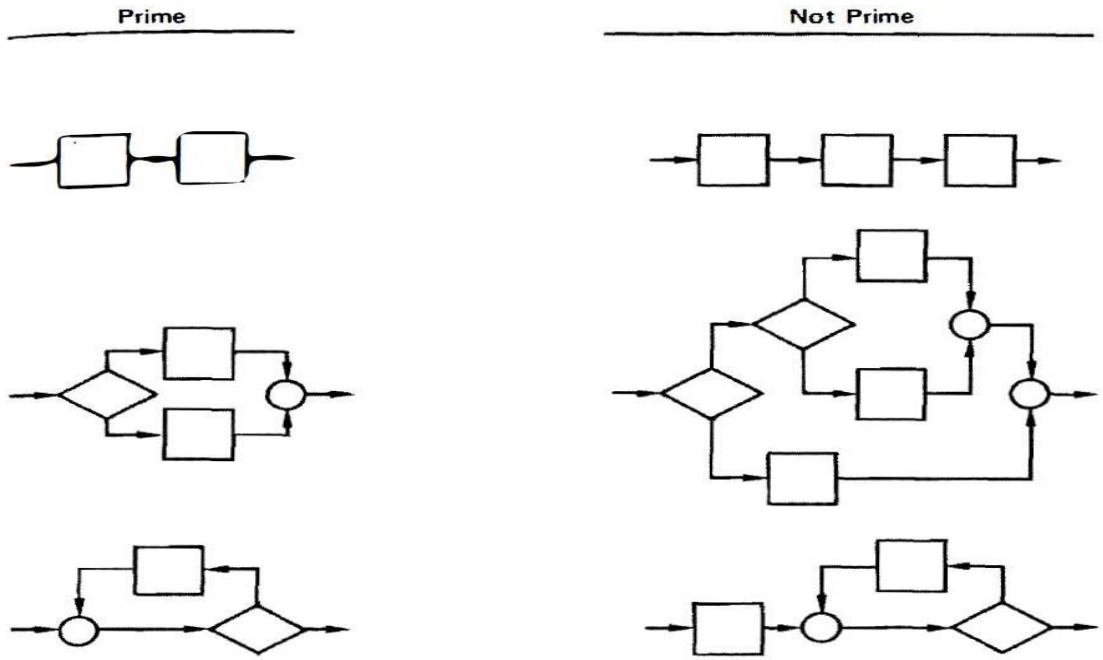
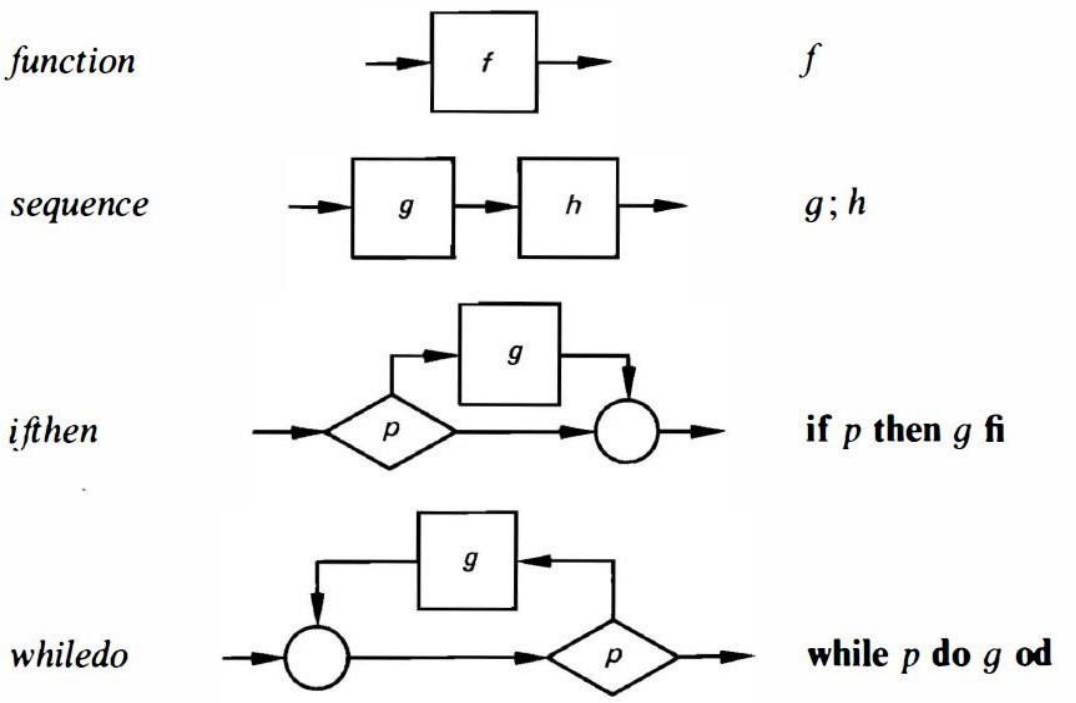


Fig. 16: Detection of prime programs.

These seven are given specific names below; they correspond to the control structures of PDL as follows:



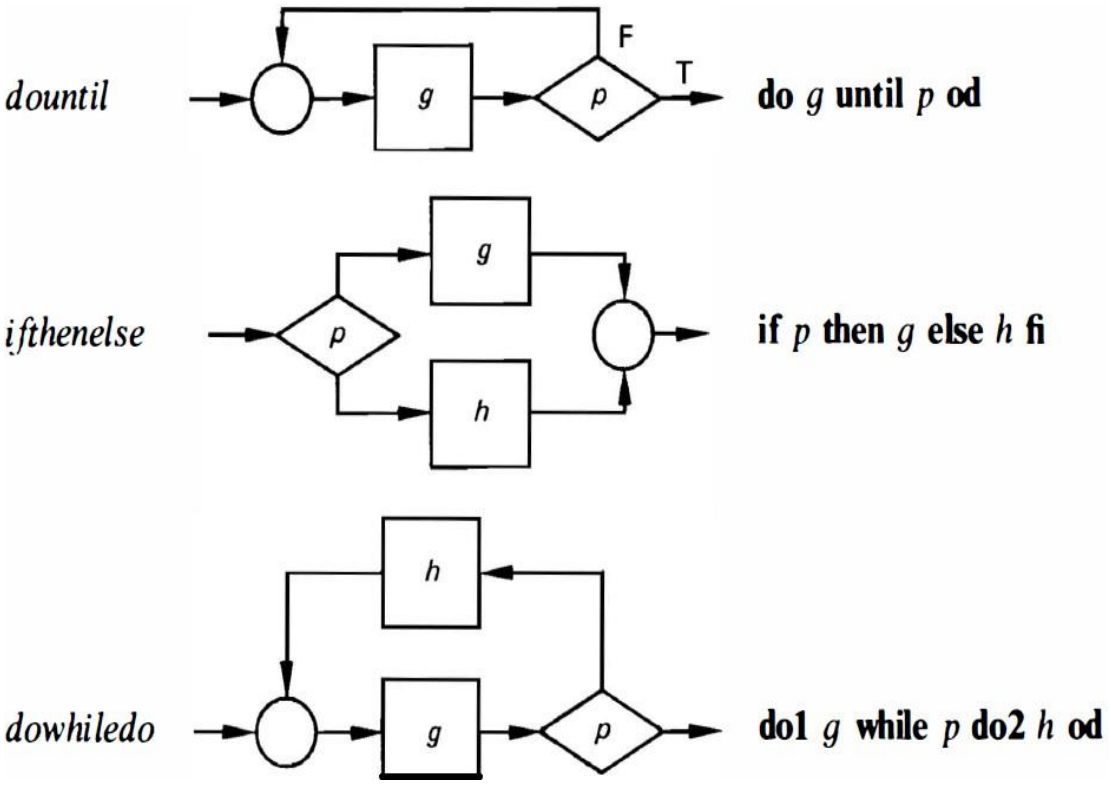


Fig.17: The control structures of PDL.

Notes:

1-

$$+(A)\,*(A) = *(A) + (A) = +(A)$$
$$*(A)\,*(A) = *(A)$$
$$A\,*(A) = *(A)A = +(A)$$
$$A\emptyset = \emptyset A = A$$
$$*(*(A)) = *(A)$$
$$A(B \mid C) = (AB) \mid (AC),\ (B \mid C)A = (BA) \mid (CA)$$
$$A(BC) = (AB)C$$
$$+(A) \mid \emptyset = *(A)$$

2.5 CPU architecture:

To fully understand how a computer can work it is needed to understand first the components of which it is built of. A processor will have:

A. Clock module

B. Registers

C. ALU

D. Output

E. Counter

F. RAM

G. Bus

A- Clock module:

- The clocks function in the computer is to cycle forward the data or instructions.
- The clock is a crucial component and without the clock module the computer will simply not work or rather not show any progress of the data.
- The clock module also determines how fast the computer can execute instructions.
- This is most commonly measured in frequency (Hz),
- The clocks output will be voltages driven outputted in pulses.
- For the simple case there is digital circuit such as the 555 Timer IC. This clock can be regulated with only a capacitor and a resistor, the values of the chosen components will determinate on how fast the clock will cycle.1
- This way the clock will work as a basic flip flop and will be always on until the circuit or the clock has been reset.

B- Registers:

- ❖ Registers (store values) are the processors component which holds values (voltages). These registers contain small amount of fast memory which hold values stored into them.
- ❖ Instructions such as mathematical numbers (written in bits) can be stored in registers and then move these numbers where they should belong.

C- ALU:

- ❖ The ALU or Arithmetic logic unit is the computer's mathematical processor.
- ❖ The ALU can input bits from different registers and then do mathematical operations when the registers values are input.
- ❖ For example on an 8-bit ALU, the ALU can only output a maximum of 256, meaning that if the ALU gets two values which both are 256 will lead to it only outputting 256.

D- Output:

- ❖ The output module of the computer consists of a register which holds a value intended for output.
- ❖ In most cases the output also consists of some graphical component which can show what kind of value the output register holds.
- ❖ Most commonly the output register is only used for the computers final instructions or result and its intents is only to show what the computer has calculated.

E- Counter:

- ❖ Using a counter or rather a program counter is crucial. The counter tells in most cases on what the computer should begin with.
- ❖ It can also be used as a indicator on what stage the computer is at. This

way it is possible to use the counter as a indicator of exactly what stage the computer is at, and also feed this information along for longer program chains.

F- RAM:

- The RAM or Random Access Memory is the computer's main storage of programs.
- There is two kinds of memory, static and dynamic RAM.
- Static RAM uses flip-flops to store the data
- While dynamic stores the data in a capacitor combined with other digital circuit.

- For simple projects static RAM is usually used since its faster, easier and more reliable than dynamic memory but can store less values and costs more. Static RAM stores data in its I/O
- Ram chips have both read and write functions. When in write mode the RAM will store the data in the I/O pins to the selected address line while in read function the RAM will output
- The data in the I/O lines from the selected address lines.

G- Bus:

- The bus of the computer is used for communications. This means that all of the computers modules can output to the bus but also input from the bus.
- Without the bus there is no way for the computer to work since without the communications the different modules can't show what they are outputting or inputting.
- The bus is usually located in the center of all the components, this way communication is assured to be closest to all the different components.

2.6 Single core computer:

Single core computer is illustrated in figure below:

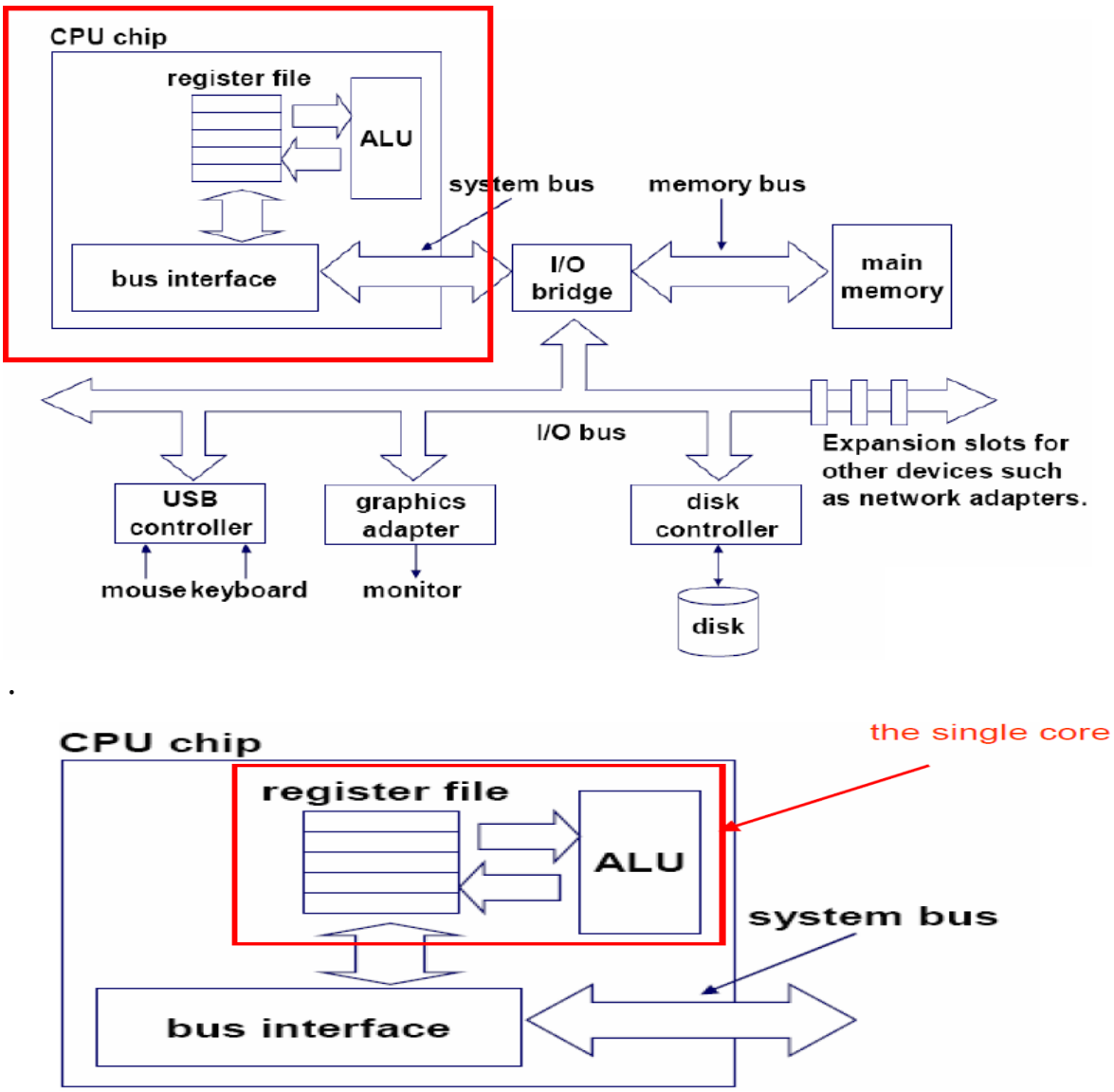


Fig.18: Single core computer.

2.7 Multi core architectures:

A new trend in computer architecture by replicating multiple processor cores on a single die as follows:

1) Multi-core processor.
2) CPU Cache.
3) CPU Coherence.

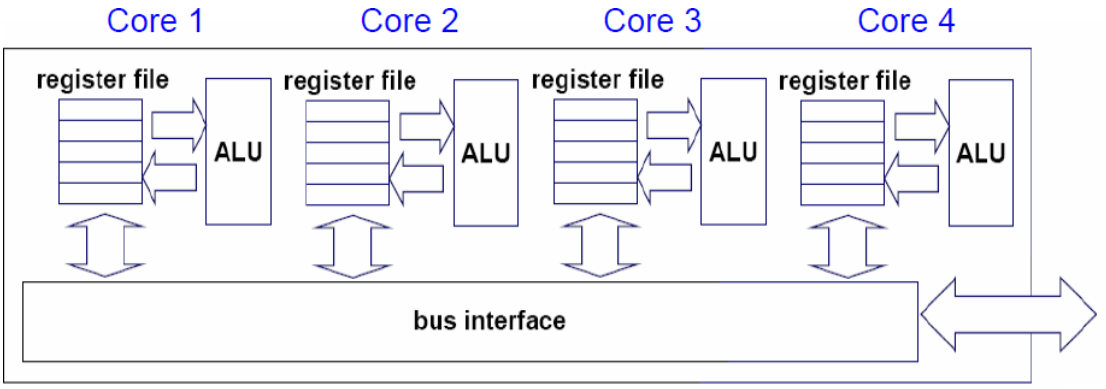


Fig.19: Multi core architectures.

A- Multi-core CPU chip:

- ❖ The cores fit on a single processor socket.
- ❖ Also, called CMP (Chip Multi-Processor) as shown in figure below:

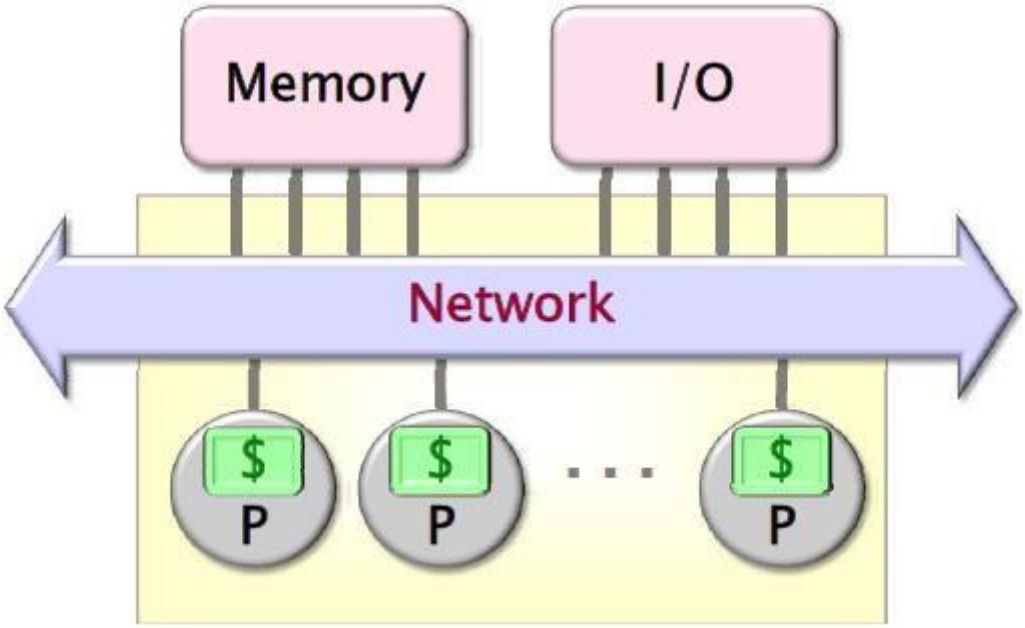


Fig.20: Chip multiprocessor (CMP).

- ❖ A multi-core processor is an integrated circuit to which two or more individual processors (called cores in this sense) have been attached.
- ❖ In a many-core processor the number of cores is large enough that traditional multi-processor techniques are no longer efficient.
- ❖ Cores on a multi-core device can be coupled tightly or loosely:
  - o may share or may not share a cache,
  - o implement inter-core communications methods or message passing.

- ❖ Cores on a multi-core implement the same architecture features as single-core systems such as instruction pipeline parallelism (ILP), vector-processing, SIMD or multi-threading.
- ❖ Many applications do not realize yet large speedup factors: parallelizing algorithms and software is a major on-going research area.

B- <u>CPU cache:</u>

- ❖ A CPU cache is an auxiliary memory which is smaller, faster memory than the main memory and which stores copies of the main memory locations that are expectedly frequently used as shown in figure below.
- ❖ Most modern desktop and server CPUs have at least three independent caches: the data cache, the instruction cache and the translation look-aside buffer.
- ❖ Each location in each memory (main or cache) has
  - o a datum (cache line) which ranges between 8 and 512 bytes in size, while a datum requested by a CPU instruction ranges between 1 and 16.
  - o a unique index (called address in the case of the main memory)
- ❖ In the cache, each location has also a tag (storing the address of the corresponding cached datum).

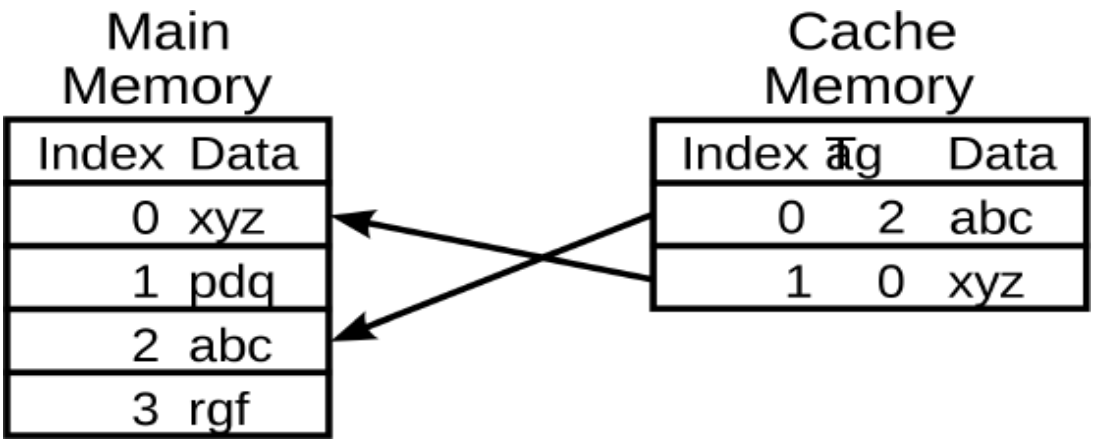


Fig.21: Main and cache memories locations.

Advantages:

1) Cache coherency circuitry, operate at higher rate than off-chip.
2) Reduced power consumption for a dual core vs two coupled single-core processors (better quality communication signals, cache can be shared)

2.8 Object oriented programming:

- Everything in an object.
- A program is a set of objects telling each other what to do by sending messages.
- Each object has its own memory (made up by other objects).
- Every object has a type.
- All objects of a specific type can receive the same messages.

The object concept:

- An object is an encapsulation of data.
- An object has identity (a unique reference) social security number (cpr), employee number, passport number.

- State, also called characteristics (variables) hungry, sad, running, alive
- Behavior (methods) eat, drink, wave, smile
- An object is an instance of an class.
- A class is often called an Abstract Data Type (ADT).

The class concept:

- A class is a collection of objects (or values) and a corresponding set of methods.
- A class encapsulates the data representation and makes data access possible at a higher level of abstraction.

General questions of chapter 2

1. Complete the following sentences:

1) …… this shows that the effect on data of any arbitrarily complex control structure can be accomplished by the use of process design language (PDL) control structures alone.

2) A …….. is a directed graph that depicts the flow of execution control of program and the instructions to be executed.

3) If the execution of a control instruction affects no data except an instruction counter, it is a ……; otherwise control instruction has …… that change data values.

4) If a flowchart node has a single in-line and single out-line, it is called a…….

5) If a flowchart node has a single in-line and two out-lines and is a pure control instruction, it is called a …….

6) It is convenient to introduce one more node with a "no-op" instruction that does not change or evaluate data, with two in-lines and one out-line, called a……

7) The …… of a flowchart preserves the ordering of function nodes, predicate nodes, and collecting nodes, but ignores the identity of associated functions, predicates, and predicate values.

8) A.....obtained by reversing each line and interchanging the roles of predicate and collecting nodes.

9) A..... is a program with a control structure that has a single entry line and a single exit line, and for each node, has a path through that node from the entry line to the exit line.

10) These execution sequences can often be better understood in terms of a

finite tree called an ……

11) If loops are not present in the flowchart, then the corresponding finite E-chart is the …….

12) If …… are present in a flowchart, the …… is the infinite tree obtained by repeatedly replacing each repeated node by the subtree beginning at the first occurrence of that node earlier in the path.

13) Equal is a ..... (7=7 is true).

14) A …… is a function whose members are ordered pairs of ordered pairs.

15) .....are useful in program design.

16) A ..... is a compact way of describing a language.

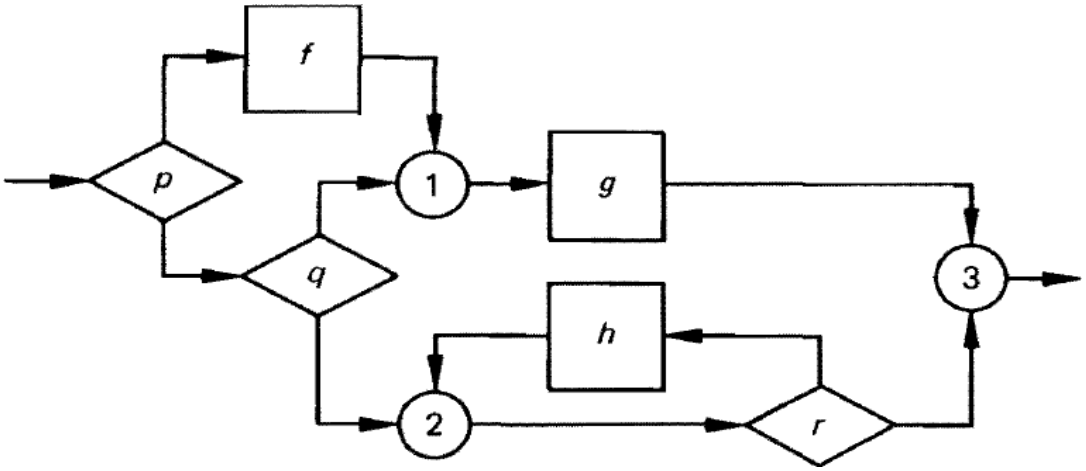
17) A …… is a proper program that has no proper subprogram of more than one node.

18) The..... determines how fast the computer can execute instructions.

19) The clocks output will be voltages driven outputted in …….

20) …… (store values) are the processors component which holds values (voltages). They contain small amount of fast memory which hold values stored into them.

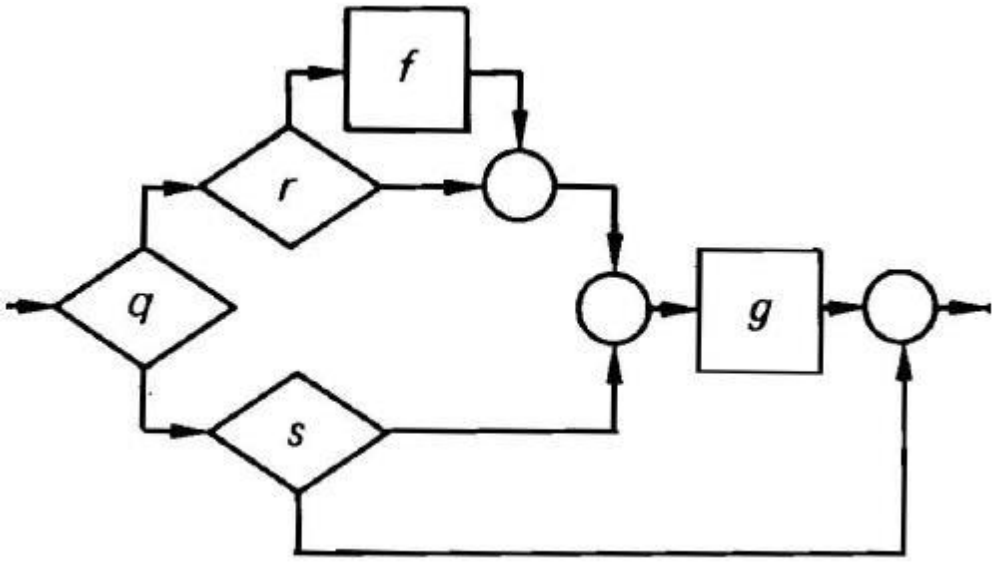
2. **Obtain the E-chart for the following flowchart program:**



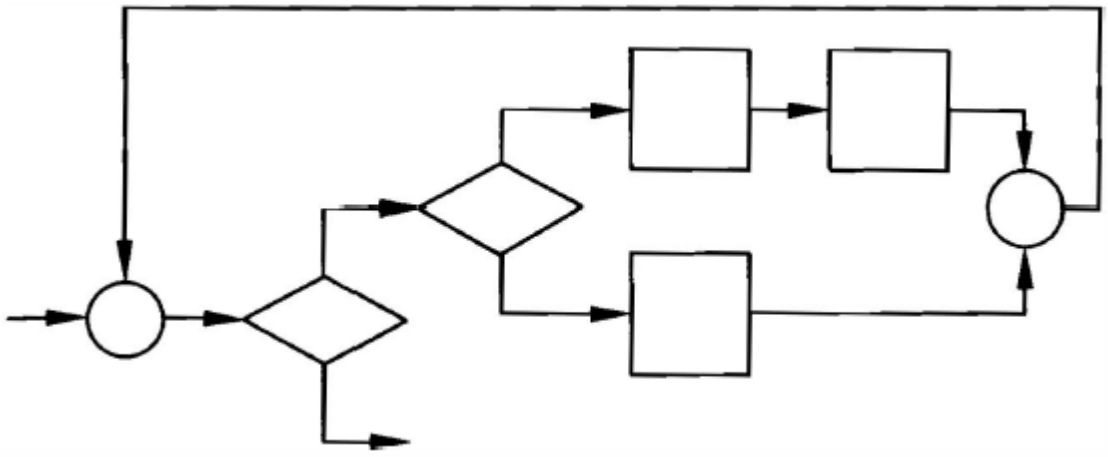
3. **Simplify the following regular expression and Draw the final simplest flowchart**?

$$((BA)|(CA))((A*(A))|\emptyset)$$

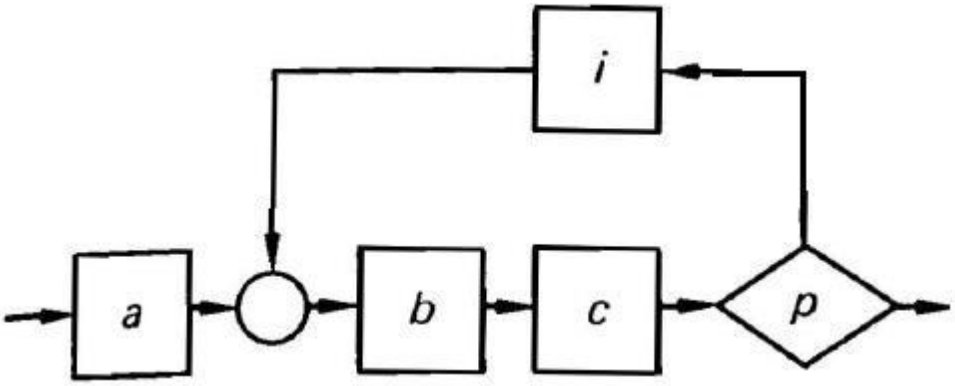
4. **Find the proper subprograms and not proper subprograms, Also Find the corresponding E-chart?**



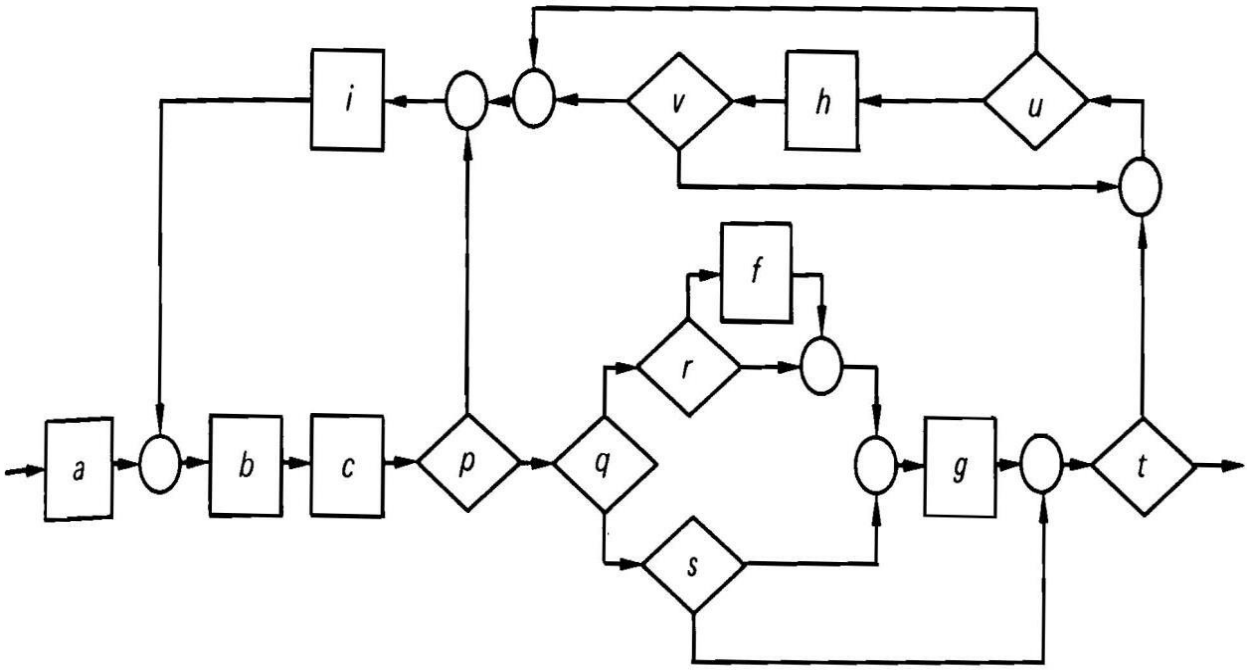
5. **Is the following program prime or not? (Show your steps)**



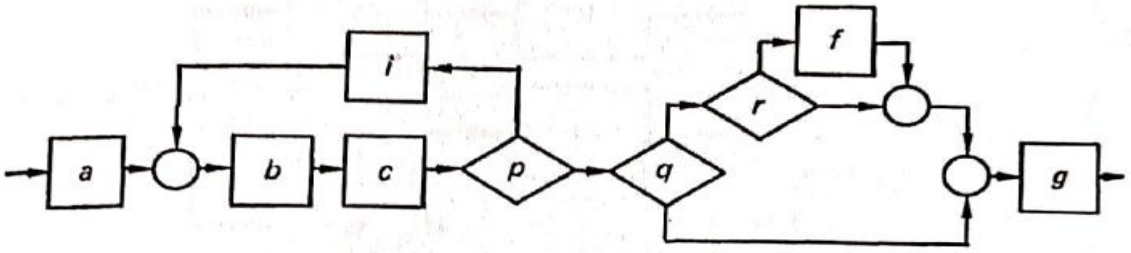
6. **Is the following program prime or not? (Show your steps)**



7. **Enumerate the proper subprograms with more than one node in the following program?**

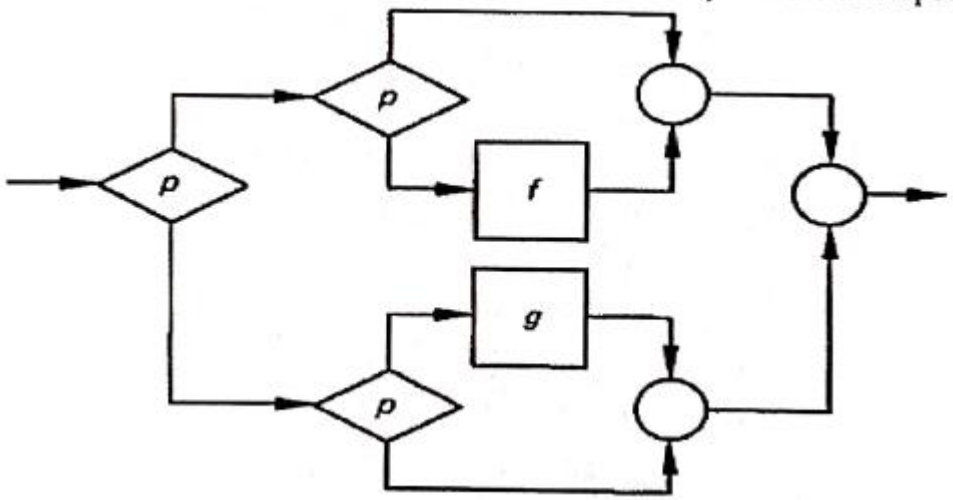


8. - Obtain the E-chart for the following flowchart program:



9.

D. The natural dual for control structure is shown below, Find the corresponding E-chart?



CHAPTER (3)

Parallel computers &

Introduction to MATLAB

CHAPTER 3: Parallel computers & Introduction to MATLAB

3.1 Parallel computers:

A- Demand for computational speed:

- Continual demand for greater computational speed from a computer system than is currently possible.
- Areas requiring great computational speed include numerical modeling and simulation of scientific and engineering problems.
- Computations must be completed within a "reasonable" time period. (Example: Weather forecasting).

B- Parallel computing:

- Using more than one computer, or a computer with more than one processor, to solve a problem.
- Usually faster computation - very simple idea - that *n* computers operating simultaneously can achieve the result *n* times faster – it will not be *n* times faster for various reasons.
- Other motives include: fault tolerance, larger amount of memory available,
- Parallel computers - computers with more than one processor – and their programming - parallel programming - has been around for more than 40 years.

C- Conventional Computer:

- Consists of a processor executing a program stored in a (main) memory as shown in figure below:

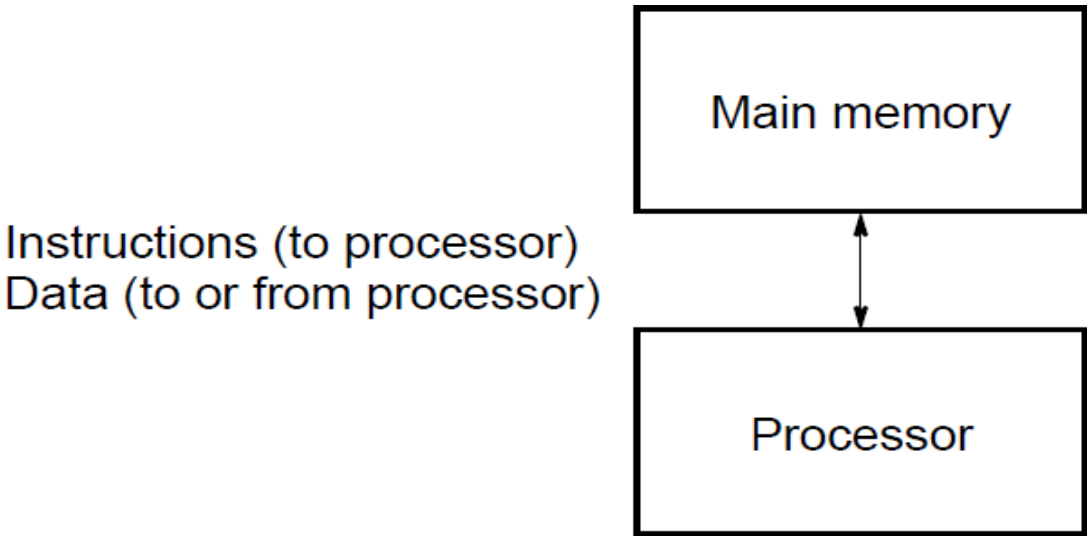


Fig. 3.1: Conventional computer.

- Each main memory location located by its address. Addresses start at 0 and extend to $(2^n - 1)$ when there are n bits (binary digits) in the address.

D- Types of parallel computers:

Two principal types:

1 Shared memory multiprocessor.

2 Distributed memory multicomputer.

1 Shared Memory Multiprocessor System:

- Natural way to extend single processor model - have multiple processors connected to multiple memory modules, such that each processor can access any memory module - so-called shared memory configuration as shown in figure below:

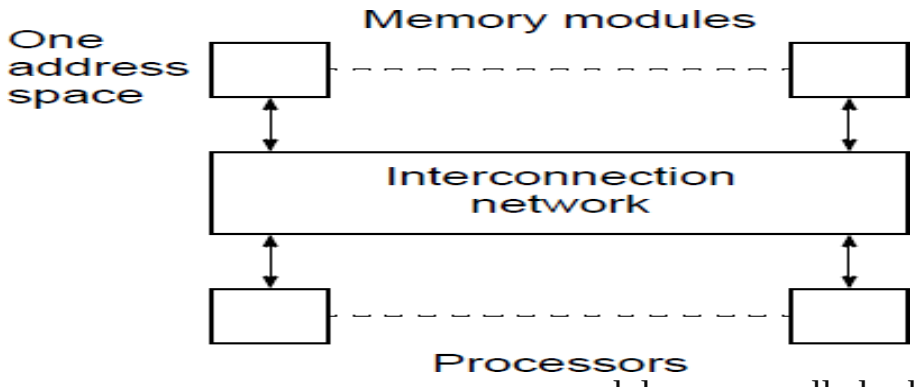


Fig. 3.2: Shared memory multiprocessor system.

- Any memory location can be accessible by any of the processors. **A single address space** exists, meaning that each memory location is given a unique address within a single range of addresses.
- Generally, shared memory programming more convenient although it does require access to shared data to be controlled by the programmer.

2 Distributed memory multicomputer:

- Message-Passing Multicomputer: Complete computers connected through an interconnection network as shown in figure below:

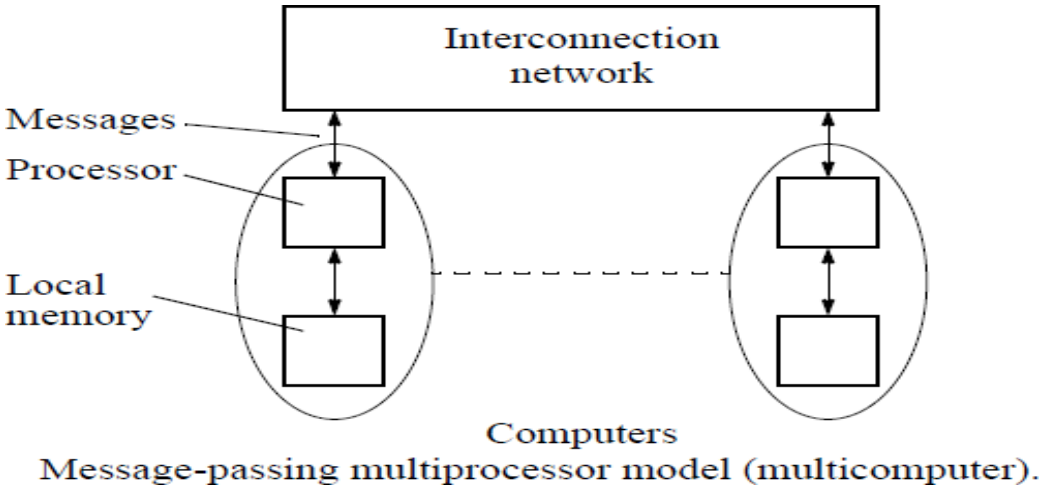


Fig. 3.3: Multicomputer system.

- Each processor has access to the whole memory using a single memory address space. For a processor to access a location not in its local memory, message passing must occur to pass data from the processor to the location or from the location to the processor, in some automated way that hides the fact that the memory is distributed as shown in figure below.

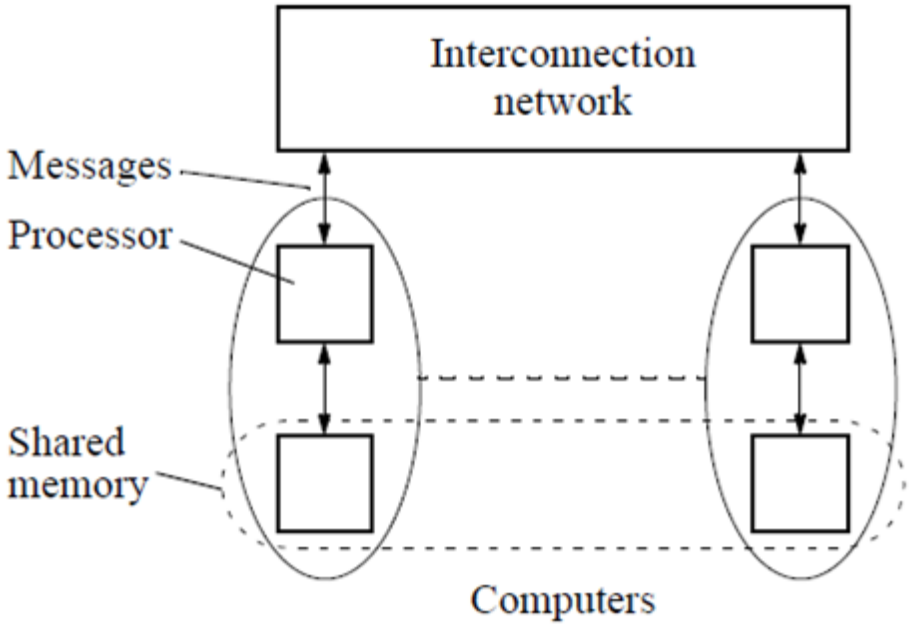


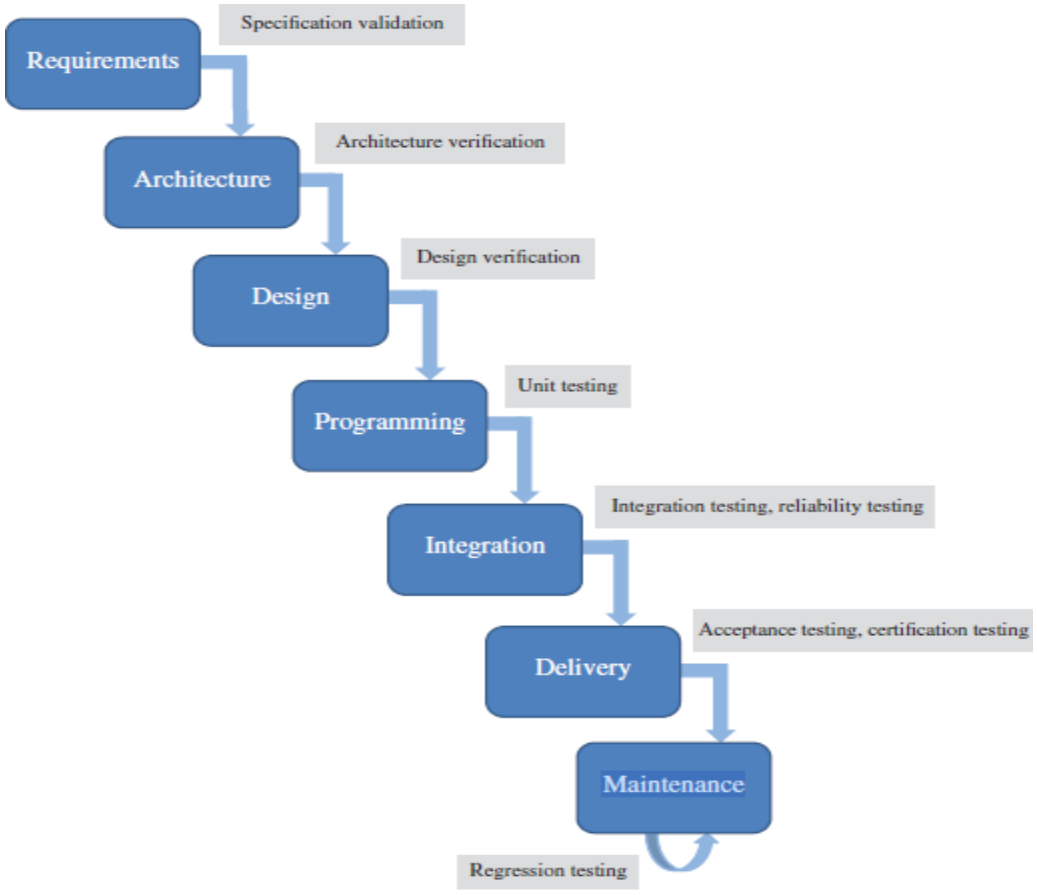
Fig. 3.4: Distributed memory multicomputer system.

3.2 Program maintenance:

It is common to distinguish in software maintenance between several types of maintenance activity; the two most important types (in terms of cost) are as follows:

1 **Corrective maintenance**, which aims to remove software faults.

2 **Adaptive maintenance**, which aims to adapt the software product to evolving requirements.



A reference software lifecycle.

3.3 Program testing:

- Testing is the process of executing a software product on sample input data and analyzing its output as shown in figure below.
- The focus of software testing is to run the candidate program on selected input data and check whether the program behaves correctly with respect to its specification.

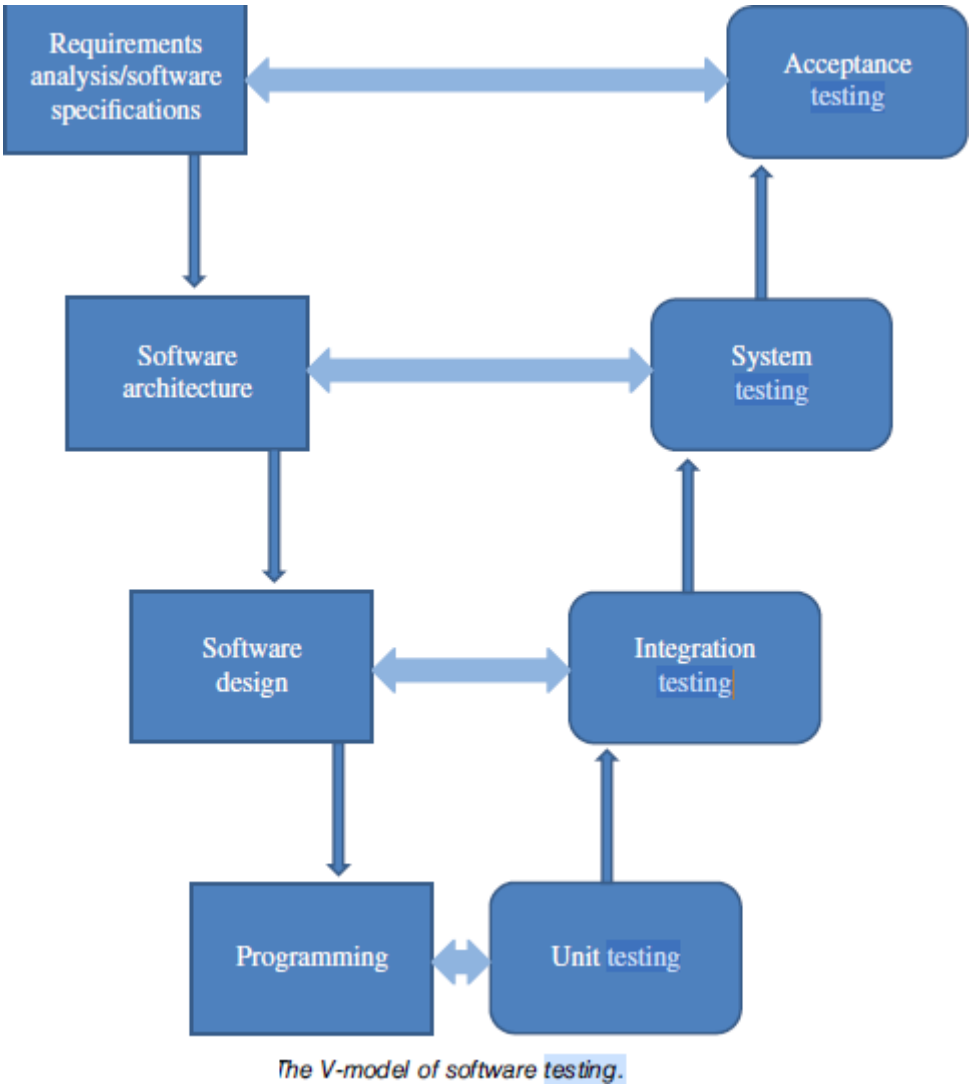


Fig. 3.5: The V-model of software testing.

- To the extent that software testing is the task of finding and removing faults in programs, the study of software testing requires that we study the nature and properties of faults.
- Testing a program to establish correctness with respect to a

specification (as in acceptance testing).

- Testing a program to ensure that it computes its intended function (as in unit testing).
- Testing a program to ensure that it is robust (i.e., that it behaves appropriately outside the domain of its specification).
- Testing a program to ensure its safety (i.e., that even when it fails to meet its specification, it causes no safety violations).

3.4 MATLAB basics:

What is Matlab?

The Matlab documentation describes Matlab as a high-performance language for technical computing, integrating computation, visualisation, and programming in an easy- to-use environment where problems and solutions are expressed in familiar mathematical notation.

MATLAB (an abbreviation of "MATrix LABoratory") and MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages.

Typical uses of Matlab include:

- Mathematics and computation;
- Algorithm development;
- Data acquisition;
- Modeling, simulation, and prototyping;
- Data analysis, exploration, and visualisation;
- Scientific, engineering and financial graphics;
- Application development, including graphical user interface building;

Matlab is used for across a wide range of application areas covering science, engineering and business/finance. Matlab is an interactive system whose basic data element is an array that does not require dimensioning. This allows you to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar non-interactive language such as C or Fortran.

1. Vectors:

Let's start by creating a vector. Enter each element of the vector (separated by a space) between brackets, and set it equal to a variable. For example, to create the vector a, enter into the MATLAB command window (you can "copy" and

"paste" from your browser into MATLAB to make it easy) the following:

```
a = [1 2 3 4 5 6 9 8 7]
```

MATLAB should return:

```
a =

     1  2  3  4  5  6  9  8  7
```

To create a vector with elements between 0 and 20 evenly spaced in increments of 2 (this method is frequently used to create a time vector), enter:

```
t = 0:2:20
t =
             0  2  4  6  8  10  12  14  16  18  20
```

Manipulating vectors is almost as easy as creating them. First, suppose you would like to add 2 to each of the elements in vector 'a'. The equation for that looks like:

```
b = a + 2
b =

    3  4  5  6  7  8  11  10  9
```

Now suppose, you would like to add two vectors together. If the two vectors are the same length, it is easy. Simply add the two as shown below:

```
c = a + b

c =

     4  6  8  10  12  14  20  18  16
```

Subtraction of vectors of the same length works exactly the same way.

2. Functions:

MATLAB includes many standard functions. Each function is a block of codes that accomplishes a specific task. MATLAB contains all of the standard functions such as ***sin, cos, log, exp, sqrt***, as well as many others. Commonly used constants such as ***pi,*** **and** ***i*** **or** ***j*** for the square root of (-1), are also incorporated into MATLAB. For example:

```
sin(pi/4)
ans =

          0.7071
```

To determine the usage of any function, type help [function name] at the MATLAB command window.

3. Plotting:

One of the most important functions in MATLAB is the plot function. Plot also happens to be one of the easiest functions to learn how to use. The basic format of the function is to enter the following command in the MATLAB command window or into an *m-file*.

```
plot(x,y)
```

This command will plot the elements of vector x on the horizontal axis of a figure, and the elements of the vector y on the vertical axis of the figure. If we wanted to plot the simple, linear formula:

y=3x

```
x = 0:0.1:10;

y = 3*x;

plot(x,y)
```

We could create an m-file with the following lines of code: which will generate the following plot as shown in figure below:

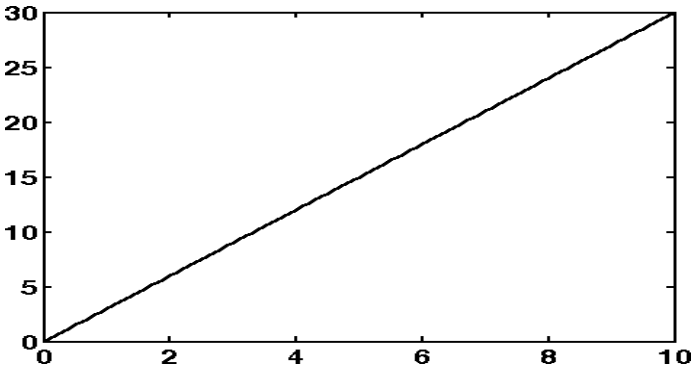


Fig.: The plot figure.

3.1 Plot Aesthetics:

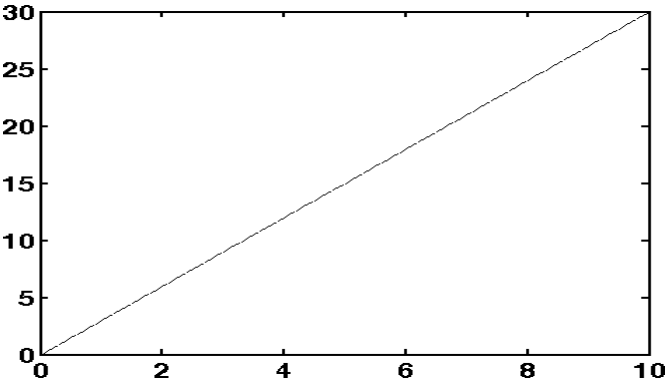
The color and point marker can be changed on a plot by adding a third parameter (in single quotes) to the plot command. For example, to plot the above function as a red, dotted line, the m-file should be changed to:

```
x = 0:0.1:10;

y = 3*x;

plot(x,y,'r:')
```

The plot now looks like as shown in figure below:



The third input consists of one to three characters, which specify a color and/or a point marker type. The list of colors and point markers is as illustrated in table below:

Table 1: color orders in Matlab mfile.

Y	**Yellow**	**.**	**point**
M	**Magenta**	**O**	**circle**
C	**Cyan**	**X**	**x-mark**
R	**Red**	**+**	**plus**
G	**Green**	**-**	**solid**
B	**Blue**	*****	**star**
W	**White**	**:**	**dotted**
K	**Black**	**-.**	**dash-dot**
		--	**dashed**

You can plot more than one function on the same figure. Let's say you want to plot a sine wave and cosine wave on the same set of axes, using a different color and point marker for each. The following m-file could be used to do this:

```
x = linspace(0,2*pi,50);

y = sin(x);

z = cos(x);

plot(x,y,'r', x,z,'bx')
```

You will get a plot of a sine wave and cosine wave as illustrated in figure below, with the sine wave in a solid red line and the cosine wave in a blue line made up of x's:

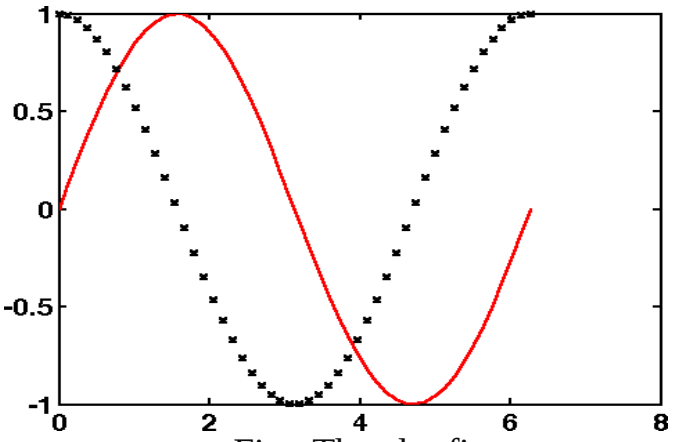


Fig.: The plot figure.

By adding more sets of parameters to plot command, you can plot as many different functions on the same figure as you want.

When plotting many things on the same graph, it is useful to differentiate the different functions based on color and point marker. The same effect can also be achieved using the hold on and hold off commands. The same plot shown above could be generated using the following m-file:

```
x = linspace(0,2*pi,50);
y = sin(x);
plot(x,y,'r')
z = cos(x);
hold on
plot(x,z,'bx')
```

3.2 Subplotting:

More than one plot can be put on the same figure using the subplot command. The subplot command allows you to separate the figure into as many plots as desired, and put them all in one figure. To use this command, the following command is entered into the MATLAB command window or an m-file:

subplot(m,n,p)

This command splits the figure into a matrix of m rows and n columns, thereby creating m*n plots on one figure. The p'th plot is selected as the currently active plot. For instance, suppose you want to see a sine wave, cosine wave, and tangent wave plotted on the same figure, but not on the same axis. The following m-file will accomplish this:

```
x = linspace(0,2*pi,50);
y = sin(x);
z = cos(x);

w = tan(x);

subplot(2,2,1)

plot(x,y)
```

you should have the following plot as shown in figure below:

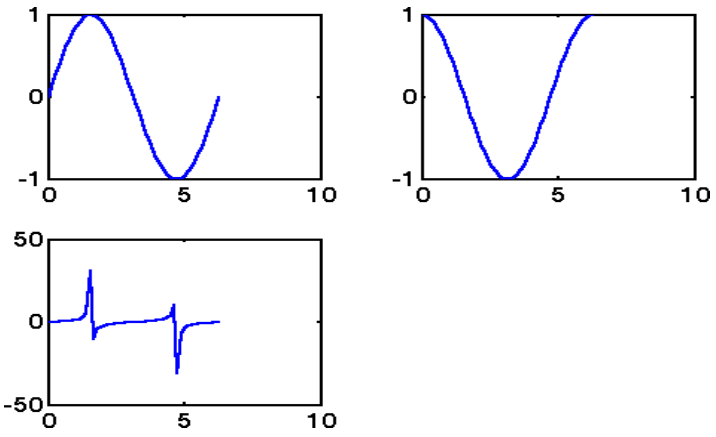


Fig.: The subplot figure.

- Now that you have found different ways to plot functions, you can customize your plots to meet your needs. The most important way to do this is with the axis command. The axis command changes the axis of the plot shown, so only the part of the axis that is desirable is displayed. The axis command is used by entering the following command after the plot command (or any command that has a plot as an output):

```
axis([xmin, xmax, ymin, ymax])
```

For instance, suppose you want to look at a plot of the function:

$y = e^{5t} + 1$ If you enter the following into MATLAB:

```
t=0:0.01:5;

y=exp(5*t)-1;

plot(t,y)
```

you should have the following plot as shown in figure below:

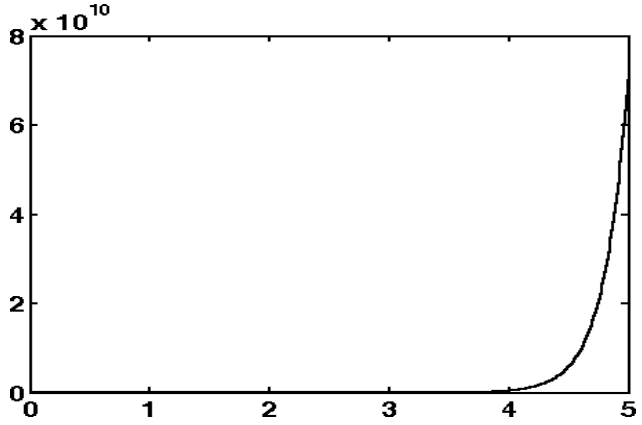


Fig.: The plot figure.

As you can see, the plot goes to infinity. Looking at the y-axis (scale: 8e10), it is apparent that not much can be seen from this plot. To get a better idea of what is going on in this plot, let's look at the first second of this function. Enter the following command into the MATLAB command window:

```
axis([0  1  0  50])
```

and you should get the following plot as shown in figure below:

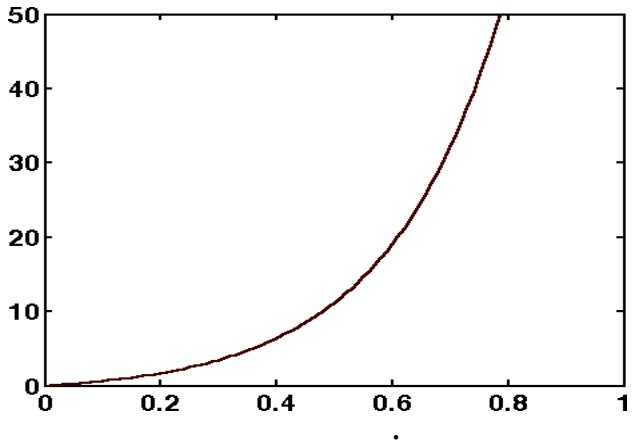


Fig.: The plot figure.

3.3 Adding text:

Another thing that may be important for your plots is labeling. You can give your plot a title (with the title command), x-axis label (with the xlabel command), y-axis label (with the ylabel command), and put text on the actual plot. All of the above commands are issued after the actual plot command has been issued.

- **A title** will be placed, centered, above the plot with the command: title (**'title string').**
- **The x-axis label** is issued with the following command: xlabel (**'x-axis string'**).
- **The y-axis label** is issued with the following command: ylabel (**'y-axis string'**).

Furthermore, text can be put on the plot itself in one of two ways: the text command and the gtext command. The first command involves knowing the coordinates of where you want the text string. The command is text (**xcor,ycor,'textstring'**).

To use the other command, you do not need to know the exact coordinates. The command is gtext(**'text string'),** and then you just move the cross-hair to the desired location with the mouse, and click on the position you want the text placed.

To further demonstrate labeling, take the step response plot from above. Assuming that you have already changed the axis, copying the following lines of text after the axis command will put all the labels on the plot:

```
title('Exponential Function')

xlabel('Time (sec)')

ylabel('Output')

gtext('Unnecessary Labeling')
```

The text "unnecessary labeling" was placed right above the position, you can clicked on. The plot should look like the following:

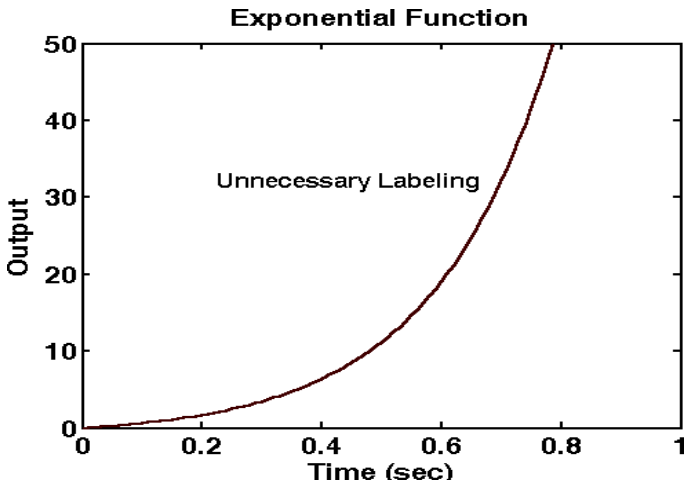


Fig.: The plot figure with xlabel, ylabel and tiltle.

Other commands that can be used with the plot command are:

- **grid(adds grid line to your plot).**

Suppose now that you want to plot a sine wave as a function of time. First, make a time vector (the semicolon after each statement tells MATLAB we don't want to see all the values) and then compute the *sin* value at each time as follows:

```
t=0:0.25:7;
y = sin(t);
plot(t,y)
title('Sine Wave as a Function of Time')
xlabel('Time (sec)')
ylabel('Output')
```

You should obtain the following plot as shown in figure below:

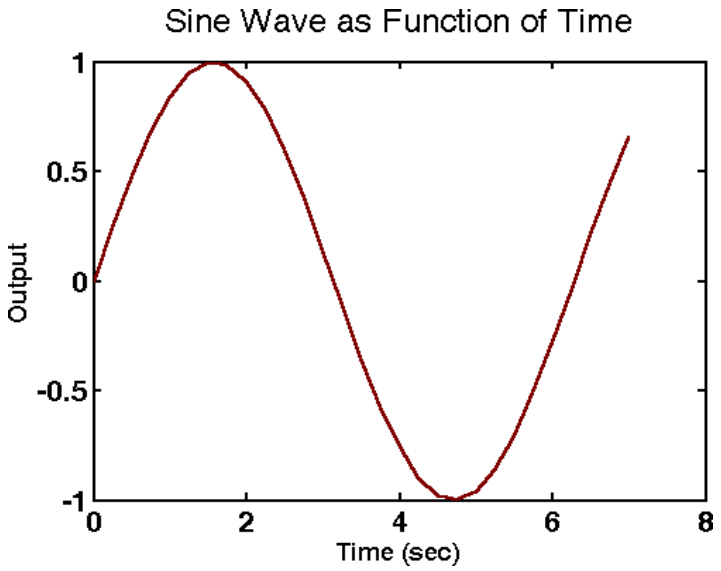


Fig.: The plot figure with xlabel, ylabel and tiltle.

This plot contains approximately one period of a sine wave. Basic plotting is very easy in MATLAB, and the plot command has extensive add-on capabilities.

4. Polynomials:

In MATLAB, a polynomial is represented by a vector. To create a polynomial in MATLAB, simply enter each coefficient of the polynomial into a vector in descending order. For instance, let's say you have the following polynomial:

$$s^4+3s^3-15s^2-2s+9$$

To enter this into MATLAB, just enter it as a vector in the following manner:

```
x = [1 3 -15 -2 9]

x =

     1  3  -15  -2  9
```

MATLAB can interpret a vector of length n+1 as an nth order polynomial. Thus, if your polynomial is missing any coefficients, you must enter zeros in the appropriate place in the vector. For example:

$$s^4+1$$

would be represented in MATLAB as:

```
y = [1 0 0 0 1]
```

You can find the value of a polynomial using the ***polyval*** function. For example, to find the value of the above polynomial at s=2:

```
z = polyval([1 0 0 0 1],2)

z =

    17
```

You can also extract the roots of a polynomial. This is useful when you have a high- order polynomial such as:

$$s^4+3s^3-15s^2-2s+9$$

Finding the roots would be as easy as entering the following command:

```
roots([1 3 -15 -2 9])

ans =

   -5.5745

    2.5836

   -0.7951

    0.7860
```

The product of two polynomials is found by taking the convolution of their coefficients. MATLAB's function ***conv*** will be used as follows:

```
x = [1 2];

y = [1 4 8];

z = conv(x,y)

z =

     1   6   16   16
```

Dividing two polynomials is just as easy. The ***deconv*** function will return the remainder as well as the result. Let's divide z by y and see if we get x.

```
[xx, R] = deconv(z,y)

xx =

     1   2

R =

     0   0   0   0
```

As you can see, this is just the polynomial/vector x from before.

If you want to add two polynomials together which have the same order, a simple z=x+y will work (the vectors x and y must have the same length).

5. Matrices:

Entering matrices into MATLAB is the same as entering a vector, except each row of elements is separated by a semicolon (;) or a return as shown:

```
B = [1 2 3 4;5 6 7 8;9 10 11 12]
B =  1   2   3   4

     5   6   7   8

     9  10  11  12
```

Matrices in MATLAB can be manipulated in many ways. For one, you can find the transpose of a matrix using the apostrophe key as follows:

```
C = B'
C =
     1   5    9
     2   6   10
     3   7   11
     4   8   12
```

It should be noted that if C had been complex, the apostrophe would have actually given the complex conjugate transpose. To get the transpose, use (.'). The two commands are the same if the matrix is not complex.

Now you can multiply the two matrices B and C together. Remember that order matters when multiplying matrices. For example:

```
D = B * C
D =
          30    70   110
          70   174   278
         110   278   446
```

Another option for matrix manipulation is that you can multiply the corresponding elements of two matrices using the (.*) operator (the matrices must be the same size to do this). For example:

```
 E = [1 2;3 4]
 F = [2 3;4 5]
G = E .* F
E =
      1   2
      3   4
F =
      2   3
      4   5
G =
      2    6
     12   20
```

If you have a square matrix, like E, you can also multiply it by itself as many times as you like by raising it to a given power as:

```
 E^3
ans =
      37    54
      81   118
```

If wanted to cube each element in the matrix, just use the **element-by-element cubing as:**

```
E.^3
ans =
       1     8
      27    64
```

You can also find the inverse of a matrix:

```
X = inv(E)
X =
      -2.0000    1.0000
       1.5000   -0.5000
```

or its eigenvalues:

```
eig(E)
ans =
      -0.3723
       5.3723
```

There is even a function to find the coefficients of the characteristic polynomial of a matrix. The "***poly***" function creates a vector that includes the coefficients of the characteristic polynomial. Example:

```
p = poly(E)
 p =
     1.0000   -5.0000   -2.0000
```

Remember that the eigenvalues of a matrix are the same as the roots of its characteristic polynomial:

```
roots(p)
ans =
      5.3723
     -0.3723
```

3.5 Algorithm and flowchart:

Algorithm:

The word “algorithm” relates to the name of the mathematician Al-khowarizmi, which means a procedure or a technique. Software Engineer commonly uses an algorithm for planning and solving the problems. An algorithm is a sequence of steps to solve a particular problem or algorithm is an ordered set of unambiguous steps that produces a result and terminates in a finite time

Algorithm has the following characteristics:

1. **Input:** An algorithm may or may not require input
2. **Output:** Each algorithm is expected to produce at least one result
3. **Definiteness:** Each instruction must be clear and unambiguous.
4. **Finiteness:** If the instructions of an algorithm are executed, the algorithm should terminate after finite number of steps

The algorithm and flowchart include following three types of control structures.

1. **Sequence:** In the sequence structure, statements are placed one after the other and the execution takes place starting from up to down.
2. **Branching (Selection):** In branch control, there is a condition and according to a condition, a decision of either TRUE or FALSE is achieved. In the case of TRUE, one of the two branches is explored; but in the case of FALSE condition, the other alternative is taken. Generally, the „IF-THEN“ is used to represent branch control.
3. **Loop (Repetition):** The Loop or Repetition allows a statement(s) to be executed repeatedly based on certain loop condition e.g. WHILE, FOR loops.

How to write algorithms:

1. **Step 1** Define your algorithms input.
2. **Step 2** Define the variables.
3. **Step 3** Outline the algorithm's operations.
4. **Step 4** Output the results of your algorithm's operations.

Flowchart:

The first design of flowchart goes back to 1945 which was designed by John Von Neumann. Unlike an algorithm, Flowchart uses different symbols to design a solution to a problem. It is another commonly used programming tool. By looking at a Flowchart one can understand the operations and sequence of operations performed in a system. Flowchart is often considered as a blueprint of a design used for solving a specific problem.

Flowchart is diagrammatic /Graphical representation of sequence of steps to solve a problem. To draw a flowchart following standard symbols are use as illustrated in table below. Also, mathematical, relational and logical operators are shown in the second table below, respectively:

Table: Graphical representations of flowchart.

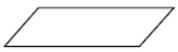
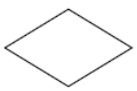
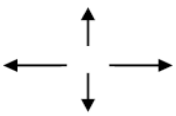
Symbol Name	Symbol	function
Oval		Used to represent start and end of flowchart
Parallelogram		Used for input and output operation
Rectangle		Processing: Used for arithmetic operations and data-manipulations
Diamond		Decision making. Used to represent the operation in which there are two/three alternatives, true and false etc
Arrows		Flow line Used to indicate the flow of logic by connecting symbols
Circle		Page Connector

Table: Mathematical, relational and logical operators.

Mathematical Operators:

Operator	Meaning	Example
+	Addition	A + B
-	Subtraction	A – B
*	Multiplication	A * B
/	Division	A / B
^	Power	A^3 for A^3
%	Reminder	A % B

Relational Operators

Operator	Meaning	Example
<	Less than	A < B
<=	Less than or equal to	A <= B
= or ==	Equal to	A = B
# or !=	Not equal to	A # B or A !=B
>	Greater than	A > B
>=	Greater tha or equal to	A >= B

Logical Operators

Operator	Example	Meaning
AND	A < B AND B < C	Result is True if both A<B and B<C are true else false
OR	A< B OR B < C	Result is True if either A<B or B<C are true else false
NOT	NOT (A >B)	Result is True if A>B is false else true

Some of selection and control statements are illustrated below:

Table: Selection control statements.

Selection control Statements

Selection Control	Example	Meaning

IF (Condition) Then
…
ENDIF
	IF (X > 10) THEN
	Y=Y+5
	ENDIF
		If condition X>10 is True execute the statement between THEN and ENDIF

IF (Condition) Then
…
ELSE
…..
 	
ENDIF
	IF (X > 10) THEN
	Y=Y+5
	ELSE
	Y=Y+8
	Z=Z+3
	ENDIF
		If condition X>10 is True execute the statement between THEN and ELSE otherwise execute the statements between ELSE and ENDIF

Loop control statements:

1 **For**:

Syntax

```
for index = values
   program statements
          :
end
```

2 **While**:

Syntax

```
while expression
   statements
end
```

3.6 Examples: Example # 1:

Algorithm & Flowchart to find the sum of two numbers

Algorithm

Step-1 Start

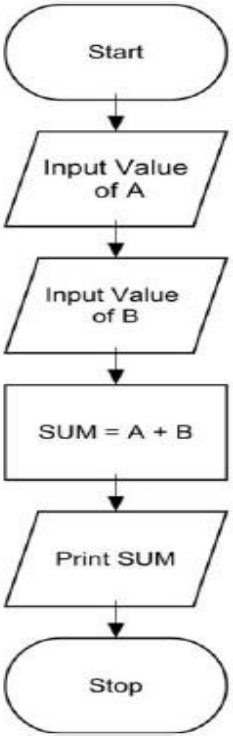
Step-2 Input first numbers say A

Step-3 Input second number say B

Step-4 SUM = A + B

Step-5 Display SUM

Step-6 Stop



Example # 2:

Algorithm & Flowchart to convert temperature from Celsius to Fahrenheit

C : temperature in Celsius
F : temperature Fahrenheit

Algorithm

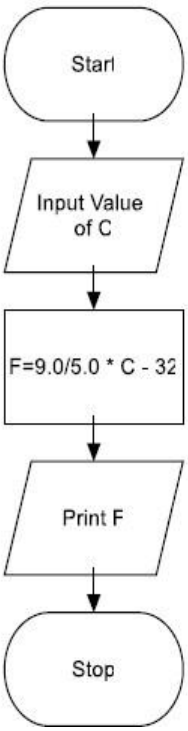
Step-1 Start

Step-2 Input temperature in Celsius say C

Step-3 F = (9.0/5.0 x C) + 32

Step-4 Display Temperature in Fahrenheit F

Step-5 Stop



Example # 3:

Algorithm & Flowchart to find Area and Perimeter of Square

L : Side Length of Square
AREA : Area of Square
PERIMETER : Perimeter of Square

Algorithm

Step-1 Start

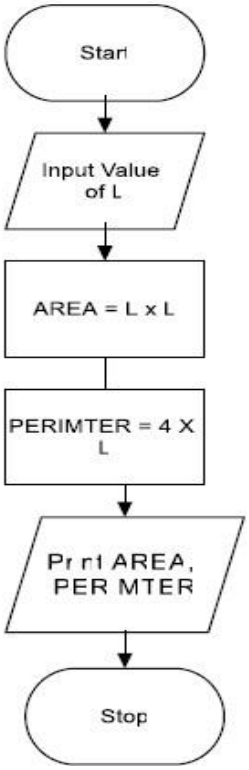
Step-2 Input Side Length of Square say L

Step-3 Area = L x L

Step-4 PERIMETER = 4 x L

Step-5 Display AREA, PERIMETER

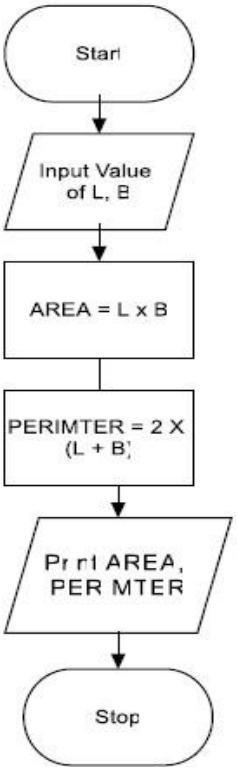
Step-6 Stop



Example # 4:

Algorithm & Flowchart to find Area and Perimeter of Rectangle

L : Length of Rectangle
B : Breadth of Rectangle
AREA : Area of Rectangle
PERIMETER : Perimeter of Rectangle



Algorithm

Step-1 Start

Step-2 Input Side Length & Breadth say L, B

Step-3 Area = L x B

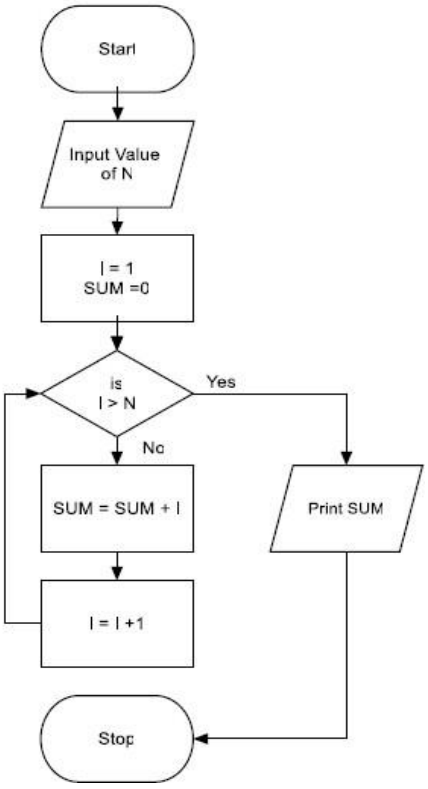
Step-4 PERIMETER = 2 x (L + B)

Step-5 Display AREA, PERIMETER

Step-6 Stop

Example # 5:

Algorithm & Flowchart to find sum of series 1+2+3+…..+N



Algorithm

Step-1 Start

Step-2 Input Value of N

Step-3 I = 1, SUM=0

Step-4 IF (I >N) THEN
GO TO Step-8
ENDIF

Step-5 SUM = SUM + I

Step-6 I = I + 1

Step-7 Go to step-4

Step-8 Display value of SUM

Step-9 Stop

Example # 6:

Algorithm & Flowchart to find sum of series $1 - X + X^2 - X^3 \ldots X^N$

Algorithm

Step-1 Start

Step-2 Input Value of N, X

Step-3 I = 1, SUM=1, TERM=1

Step-4 IF (I >N) THEN
GO TO Step-9
ENDIF

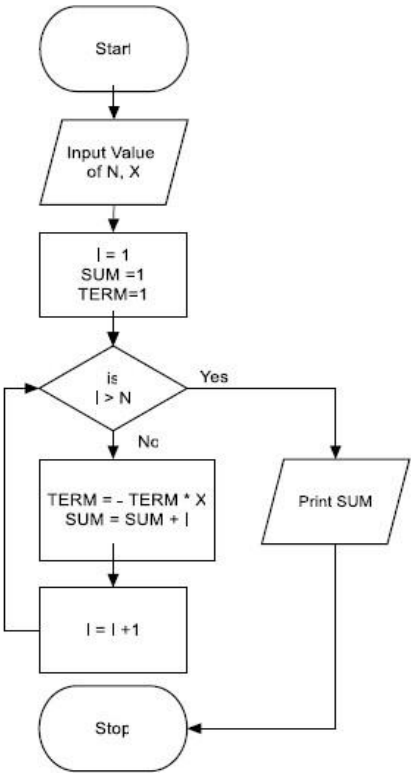
Step-5 TERM = - TERM * X

Step-6 SUM = SUM + TERM

Step-7 I = I + 1

Step-8 Go to step-4

Step-9 Display value of SUM



Example # 7

Algorithm & Flowchart to find Factorial of number n (n!=1x2x3x…n)

Algorithm

Step-1 Start

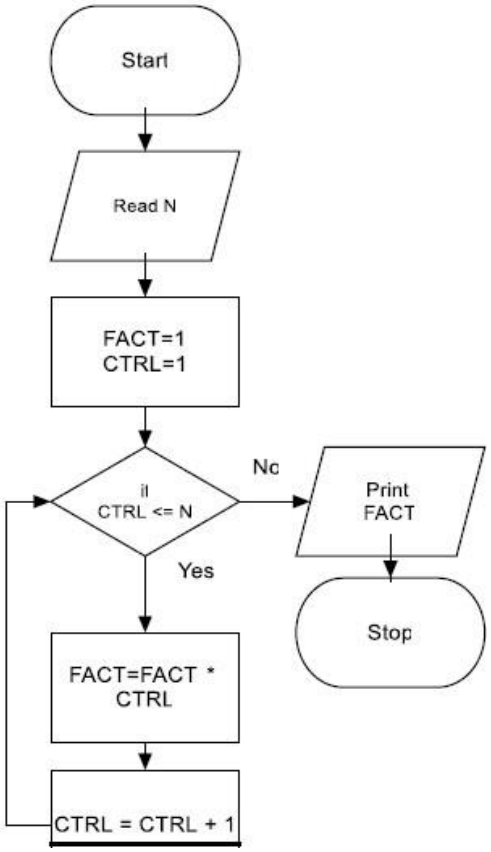
Step-2 Read number N

Step-3 FACT=1 CTRL=1

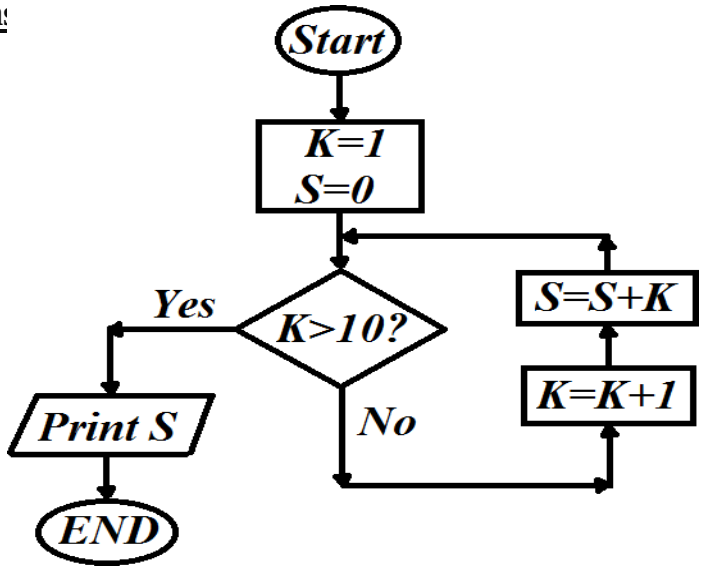
Step-4 WHILE (CTRL <= N)
DO
FACT=FACT*I
CTRL=CTRL+1
DONE

Step-5 Display FACT

Step-6 Stop



Example 8: Select the correct an



1 **For the flowchart shown, the program output is:**

A. 45

B. 66

C. 55

D. <u>Other</u>

2 **At the end of the flowchart, the value of K will be:**

A. 9

B. 10

C. <u>11</u>

D. Other

3 **If the statement K=K+1 is changed to K=K+2, the program output will be:**

A. 25

B. <u>35</u>

C. 36

D. Other

CHAPTER (4)

Software documentation

CHAPTER 4: Software documentation

4.1 Software documentation:

It is stated that any document (hardcopy or softcopy) which defines the functional requirements, design, implementation, operation or support arrangements that pertain to a software item or any data used by a software item is known as software documentation.

4.2 What is GUI:

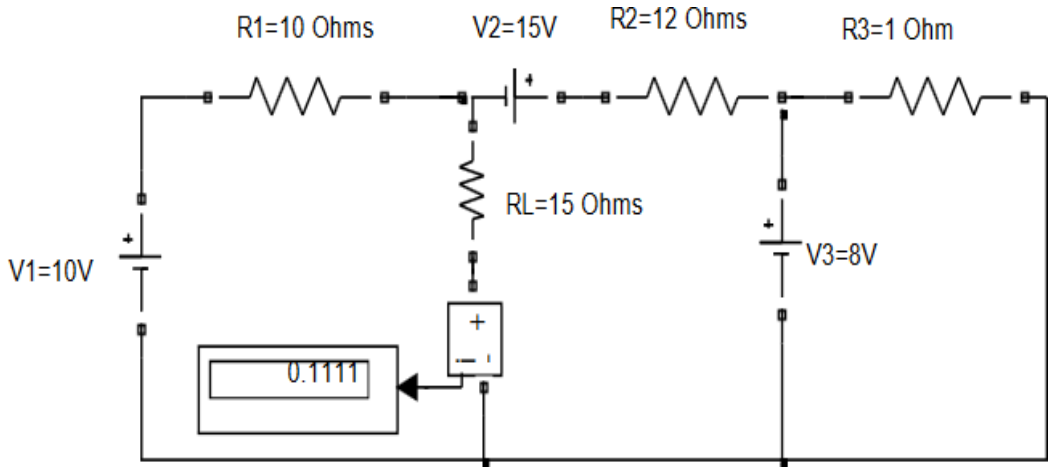
A graphical user interface (GUI) is a graphical display that contains devices, or components, that enable a user to perform interactive tasks.

4.3 What is Simulink:

Simulink is a graphical extension to MATLAB for modeling and simulation of systems. One of the main advantages of Simulink is the ability to model a nonlinear system, which a transfer function is unable to do. Another advantage of Simulink is the ability to take on initial conditions.

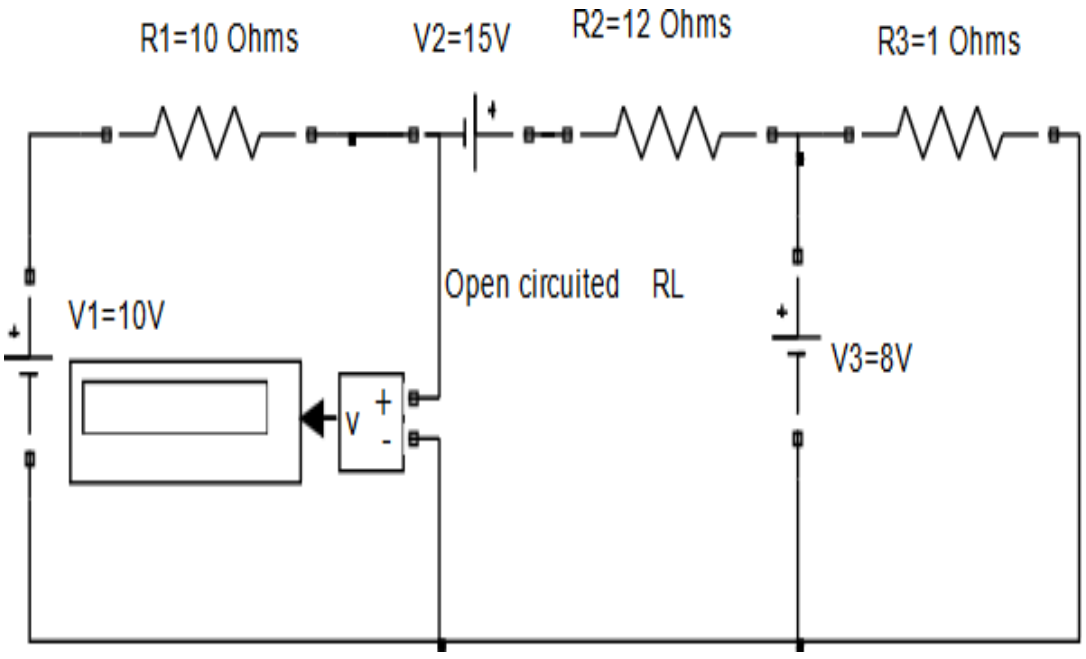
Example#1:

Find the current in branch RL=15 ohms in Matlab/Simulink ?



Example#2:

Find the voltage between V2 dc supply and R1 in Matlab/Simulink?



Example#3:

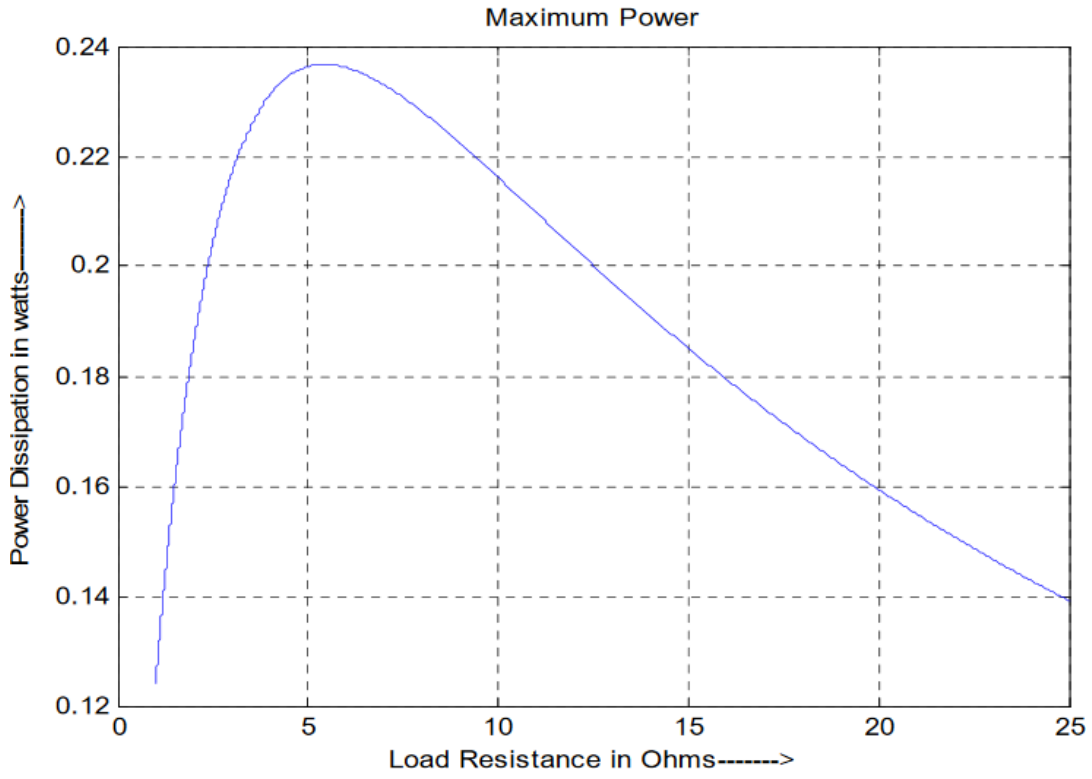
Find the error in the following program?

```
clc;
close all;
clear all;

v=input('Enter the Voltage in Volts :');
rth=input('Enter the value of Thevenins Resistance:');
rl=1:0.0001:12;
i=v./(rth+rl);
p=i.^2.*rl;
plot(rl,p);
grid;
title('Maximum Power');
xlabel('Load Resistance in Ohms------->');
ylabel('Power Dissipation in watts-------->');
```

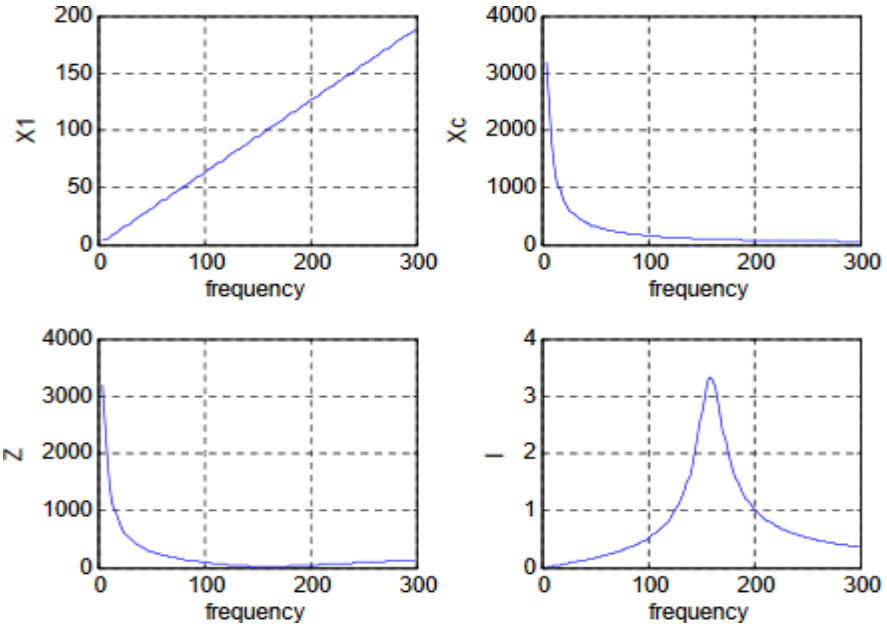
Example#4:

Find the used keywords in Matlab M-file to obtain this curve?



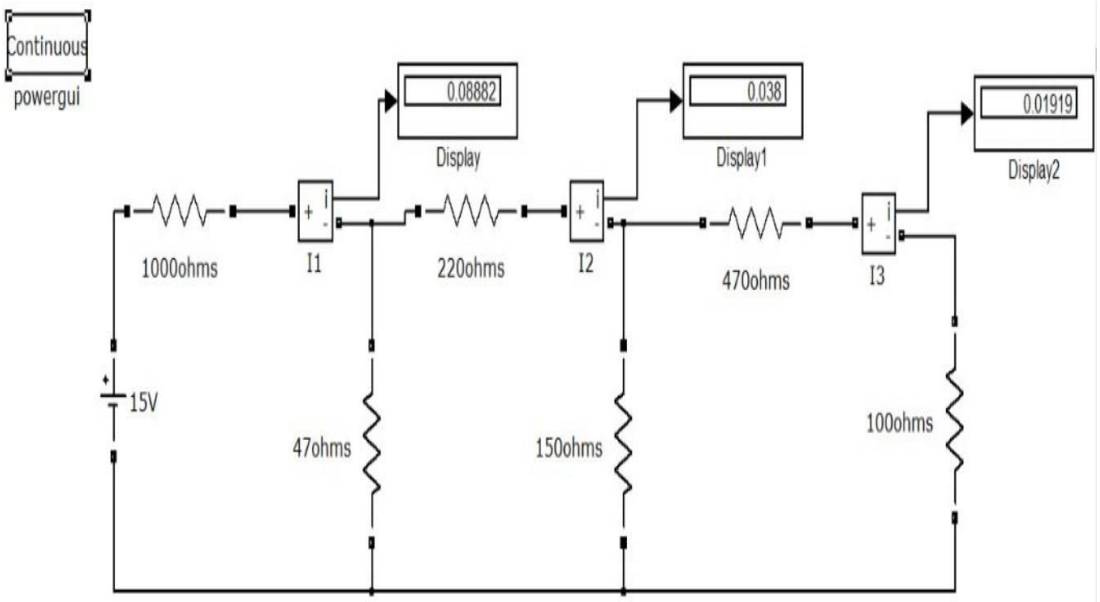
Example#5:

Find the used keyword in Matlab M-file to obtain this curve?



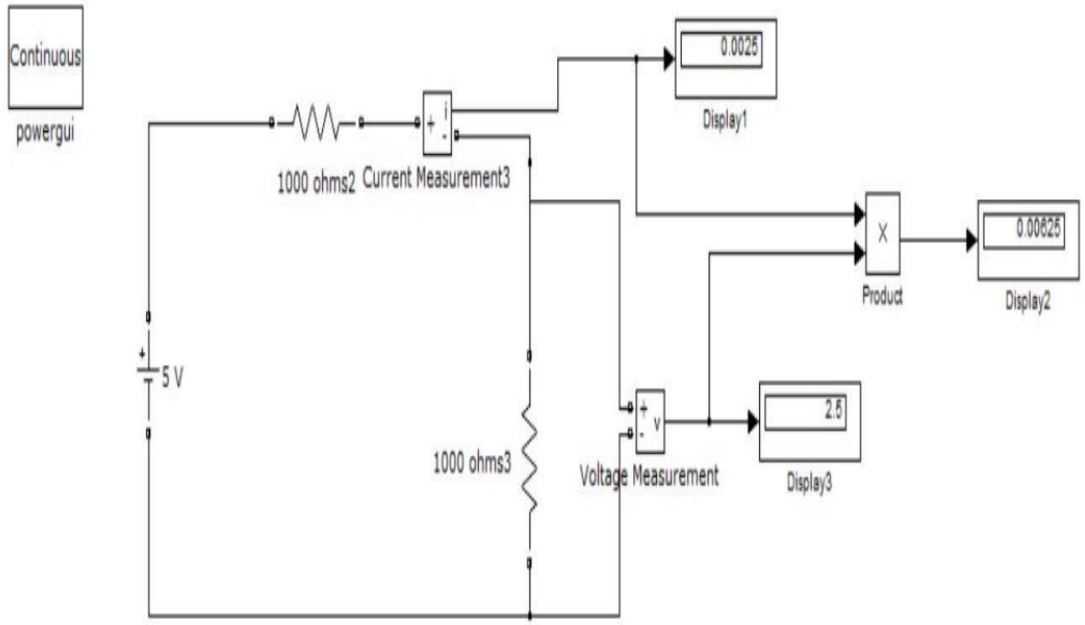
Example#6:

Find the current in the shown branches in Matlab/Simulink?



Example#7:

Find the max power transfer in Matlab/Simulink?



General questions of chapter 3,4

1) Select the correct answer:

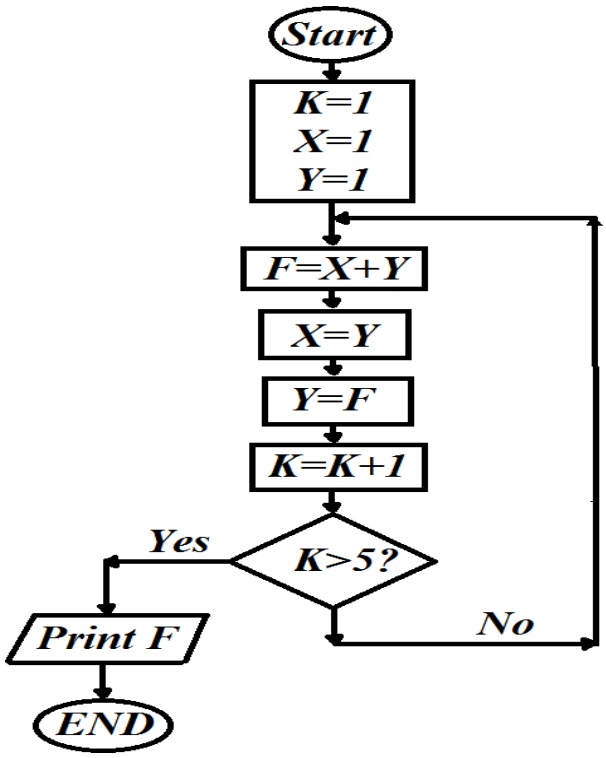
1 For the flowchart shown, the program output is:

A. 13

B. 8

C. 21

D. Other



2 At the end of the flowchart, the value of X will be:

A. 5

B. 8

C. 3

D. Other

3 To make the program print 2 at the end of the flowchart, the condition K>5 should be changed to:

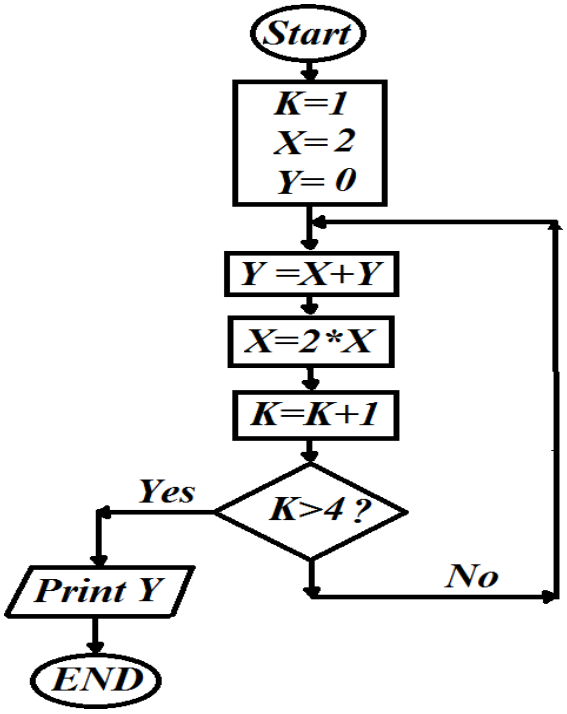
A. K< >5

B. K>1

C. K<5

D. All of the previous

2) Select the correct answer:



1. **For the flowchart shown, the program output is:**

A. 62

B. 14

C. 30

D. Other

2. **At the end of the flowchart, the value of X will be:**

A. 16

B. 32

C. 8

D. Other

3. **To make the program print 2 at the end of the flowchart, the condition K>4 should be changed to:**

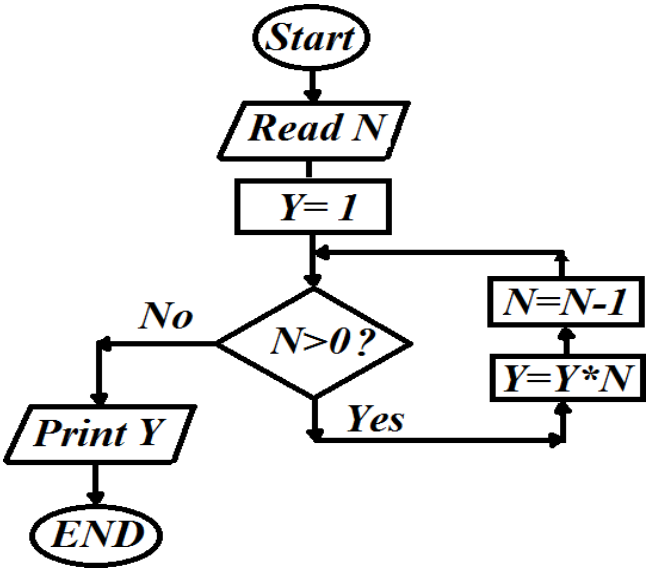
A. K< >4

B. K>1

C. K<4

D. All of before

3) Exercise: for the shown flowchart, answer the following questions:



1 **What is the program output if the input is N=2.5?**

2 **What value of N will make the program output 1?**

3 **If the condition (N>0) is changed to (N>=0), what will be the program output for the input N=4?**

4 **For the shown flowchart, if the statement N=N-1 is changed to N=N-2, What will be the output for the input N=5?**

4) Draw a flowchart for a program that calculates:

$$sum = 1 + x + x^2 + x^3$$

5) Draw a flowchart for a program that calculates:

$$sum = \left[\frac{88}{85}\right]^5 + \left[\frac{80}{77}\right]^5 + \left[\frac{72}{69}\right]^5 + \left[\frac{64}{61}\right]^5$$

6) Draw a flowchart for a program that calculates:

$$avg = \frac{3+2+1}{3}$$

❖ ***List of references:***

1-David Reed, "A Balanced Introduction to Computer Science", Prentice Hall , ISBN :013046709X, 2004.

2-Max Hailperin, Barbara Kaiser, and Karl Knight Paperback, "An Introduction to Computer Science Using Scheme" ISBN0-534-95211-9, 1999

3-Robert L. Read , "How to be a Programmer: A Short, Comprehensive, and Personal Summary", 2003

4-Introduction to Computers and Programming , *http//* citeseerx.ist.psu.edu/viewdoc/ download? doi=10.1.1.184...

5- ICDL materials,