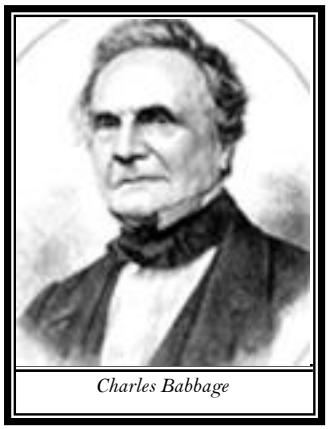
Chapter

9

Computing and Binary Operations

CONTENTS



Charles Babbage

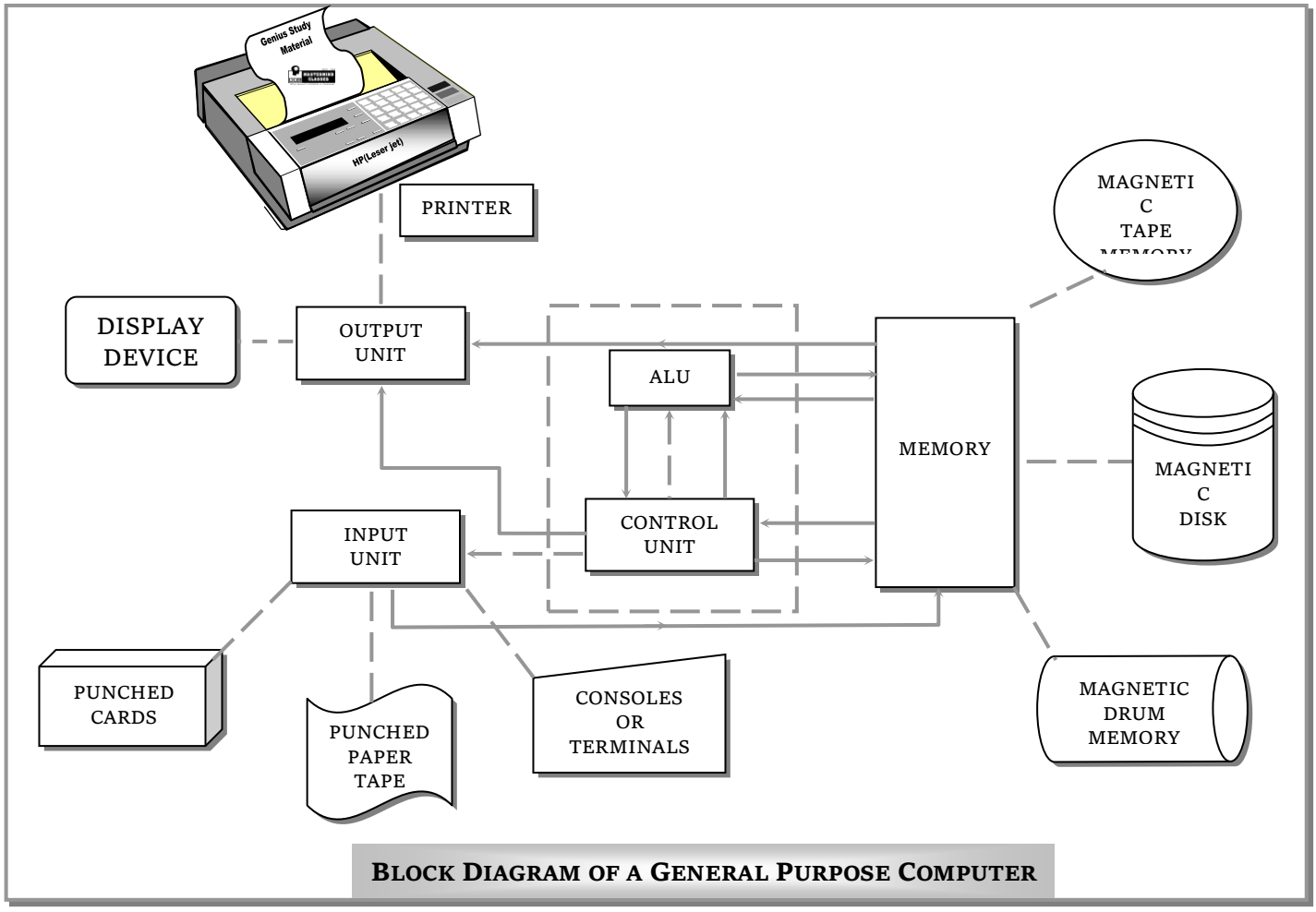
H*istorically computer work human clerks who calculated in accordance with effective methods. The term computing machine used increasingly from the 1920s, refers to any machine that does the work of a human computer* , i.e., *any machine that calculates in accordance with effective methods. During the elate 1940s and early 1950s, with the advent of electronic computing machines, the phrase 'computing machine' gradually gave way simply to computer, initially usually with the prefix electronic or digital. This entry surveys the history of these machines .*

C*harles Babbage was Lucasian professor of Mathematics at Cambridge University from 1828 to 1839 (a post formerly held by Isaac Newton). Babbage's proposed difference engine was a special-purpose digital computing machine for the automatic production of mathematical tables (such as logarithm tables, tide tables, and astronomical tables.*
Babbage exhibited a small working model in 1822.
The Swedes George and Edvard Scheutz constructed a modified version of Babbage's Difference engine.

9.1 Computing

9.1.1 Introduction

The modern digital computer or simply a computer is a general purpose electronic machine which can process a large amount of information at a very high speed. A computer can perform millions of computations in a few minutes. It can also perform arithmetical and logical operations.



BLOCK DIAGRAM OF A GENERAL PURPOSE COMPUTER

A computer has five major components :

(1) Input unit (2) Memory unit (3) Control unit

(4) Arithmetic logical unit (5) Output unit

(1) **Input unit :** The input unit is the means where the user communicates data or information to the computer.

(2) **Memory unit :** The memory unit stores instructions, data and intermediate results. It supplies, when required, the stored information to the other units of the computer.

(3) **Control unit :** The control unit controls all the activities in the computer by sending electronic command signals to other components of the computer.

(4) **Arithmetic Logical Units (ALU) :** ALU is the unit where the arithmetic and logical (*e.g.,* less than, greater than) computations are carried out. Control unit and ALU taken together is called Central Processing Unit (CPU).

(5) **Output unit :** The output unit receives the stored result from the memory unit converts it into a form. The user can understand and produces it in the desired format.

A computer may have more than one input and output units. For example, printer and display screen are two different output units attached to the same computer.

9.1.2 Memory

Our aim is to see how we can use the computer to solve some problems. For that purpose, it is useful to know a little more about main memory. From the users point of view, main memory can be thought of as a collection of compartments (or locations) as shown in fig. (i) Each compartment is assigned a number called its address (starting with zero as shown in the fig. (ii). The total number of compartments gives us the size of the memory.

0	1	2
3	4	

fig. (i) Main memory as a collection of compartments (locations)

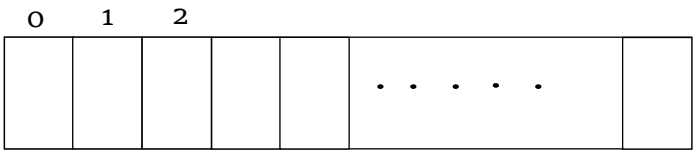


fig. (ii) Bits in a memory

Each compartment of memory (as well as a register in ALU) consists of sub-compartments fig. (ii). Each sub-compartment can store either a zero or a 1. Any information to be stored inside a computer is put using zeros and 1's. The digits 0 and 1 are called binary digits (bits in short). The acronym bit is formed by taking the letter *b* from the word 'binary' and the letters *i*, *t* from the word 'digit'. Similarly, we have the acronym dit for decimal digit, hit for hexadecimal digit etc. The number system that uses only two digits is called binary number system. Computers use binary number system for computation.

9.1.3 Algorithms

An algorithm is defined as a finite set of rules, which gives a sequence of operations for solving a specific type of problem.

In other words, algorithm is a step-by-step procedure for solving problems.

An algorithm has following five important features:

(1) Finiteness (2) Definiteness (3) Completeness (4) Input and (5) Output

(1) **Finiteness:** An algorithm should always terminate after a finite number of steps.

(2) **Definiteness:** Each step of algorithm should be precisely defined. This means that the rules should be consistent and unambiguous.

(3) **Completeness :** The rules must be complete so that the algorithm can solve all problems of a particular type for which the algorithm is designed.

(4) **Input :** An algorithm has certain inputs.

(5) **Output :** An algorithm has certain outputs which are in specific relation to the inputs.

An important consideration for an algorithm concerns its efficiency. Some algorithms are far more efficient than others in that, when programmed, one may require fewer steps or perhaps less memory than another and will therefore, be more satisfactory or economical in actually solving problems on a computer. We shall often deal with considerations of this type in the subsequent work.

In the development of an algorithm, sequence, selection and repetition (or interaction) play an important role.

(1) **Sequence :** Suppose that we want to find the value of the expression $a^3 + 4ab + b^2$ for given values of *a* and *b*. Algorithm (*i.e.,* step by step procedure) for achieving this will consist of steps given in fig. to be car

1. Get the value of *a*
2. Get the value of *b*
3. Calculate a^3, call it *S*
4. Calculate $4ab$, call it *T*
5. Calculate b^2, call it *V*
6. Find the sum $S + T + V$, call it *M*

Steps of an algorithm to evaluate $a^3 + 4ab + b^2$, given the

This algorithm, you will agree, is very straightforward, consisting of simple steps which are to be carried out one after the other. We say that such an algorithm is a *sequence* of steps, meaning that

(i) At a time only one step of the algorithm is to be carried out.

(ii) Every step of the algorithm is to be carried out once and only once; none is repeated and none is omitted.

(iii) The order of carrying out the steps of the algorithm is the same as that in which they are written.

(iv) Termination of the last step of the algorithm indicates the end of the algorithm.

Here afterwards we shall follow the convention that (i) the successive steps in a sequence will be written on successive lines and hence (ii) steps will not be necessarily numbered as they are in fig.

(2) **Selection:** An algorithm which consists of only a sequence, is not sufficient for solving any type of problem. Let us consider the problem of solving an equation of the type $m + nx = r$ (where *m, n, r* are given integers) for integral values of *x*. We immediately use laws of algebra to find $x = (r - m) \div n, n \neq 0$. Let us call an algorithm that works for only some (not necessarily all) possible sets of input values, a *semi-algorithm*.

Semi-algorithm (for the above problem) :

Step 1: Get the values of *m*, *r* and *n*.

Step 2: Subtract *m* from *r*, call this difference *b*.

Step 3: Divide *b* by *n*; print this result as the value of *x*.

The above steps are certainly efficient, As an example, let $m = 9, n = 5$ and $r = 24$, in which case we have $9 + 5x = 24$. Then in step 2, we have $24 - 9 = b$ *i.e.,* $b = 15$ and in step 3, we have $\frac{b}{n} = \frac{15}{5} = 3$, and so we print *x* = 3.

The above steps have two fatal flaws, however. First, if *n* equals 0, then either $m = r$ and *x* can have any integral value, or $m \neq r$ and no solution is possible *i.e.,* there is no integer *x* which may satisfy the given equation. Second, if there is a non-zero remainder when *b* is divided by *n* then again there is no integer *x* which may satisfy the given equation. So we must modify our algorithm to deal with all such situations as may arise. Given below is the modified algorithm which suits all the possible situations that may arise.

Step 1 : Get the value of *m*, *n* and *r*

Step 2 : **If** $n = 0$ and $m = r$

then go to step 7

else go to step 3

Step 3 : **If** $n = 0$ and $m \neq r$

then go to step 6

else go to step 4

Step 4 : **Subtract** *m* from *r*, call this difference *b* (*i.e.,* $b = r - m$)

Step 5: **Divide** *b* by *n*;

If there is a remainder

then go to step 6

else print the value of $\frac{b}{n}$, which is the

The above algorithm provides the person or computer that will execute the algorithm with an ability to choose the step to be carried out depending upon the values of *m*, *n* and *r* (and subsequently, the value of *b*). This ability is called *selection*. The power of selection is that it permits that different paths could be followed, depending upon the requirement of the problem, by the one who executes the algorithm.

In the above algorithm, selection is expressed by using the special words 'if, 'then', 'else'. Further, it may be noted that all that is written using these special words (once) constitutes one step. Note the way it is written. Nothing appears below the word 'if' till that step is over. This is known as *indentation*. The words 'then' and 'else' come with exactly same indentation with respect to the word 'if'.

(3) **Iteration or Repetition :** In forming an algorithm certain steps are required to be repeated before algorithm terminates after giving an answer. This is known as iteration or repetition.

Let us consider the problem of finding the just prime number greater than a given positive integer. The following list of steps shows the step by step procedure to be followed for solving the problem.

Consider the given integer
S { add 1 to it
test new number for primen ess
if it is prime,
then write it down and stop
S { **else** add 1 to it
test new number for primen ess
if it is prime,
then write it down and stop
S { **else** add 1 to it
test new number for primen ess
if.....

Algorithm for finding a prime number greater then a given positive integer

We see in the above procedure that the steps

"add 1 to it
test new number for primeness
if it is prime
then write it down and stop "

are repeated again and again till (after a finite number of repetitions) we get a prime number and print it. If this sequence (which involves a decision also) is denoted by S, then S is repeated again and again till, we get the result and print the result. This is technically known as *iteration or repetition*. The way of writing adopted in fig. poses a problem as we do not know the number of times *S* is repeated. This number depends upon the given positive integer. The difficulty presented above is overcome by introducing a new way of writing iterations in algorithms. The algorithm shown in fig. is (in new ways) then written as shown below

```
Consider the given
number
repeat
        add 1 to it
until the new number is
```

Or

```
Consider the given number
add 1 to it
while the new number is not
prime
   do add 1 to it
```

Two different ways of writing iteration occuring in fig.

9.1.4 Pseudo language

The languages used by human beings for talking and writing among themselves are called natural languages. Expression in a natural language can be ambiguous.

Computer, being a machine, requires that there should be no ambiguity at all when we give instructions to it. Languages used to communicate with a computer are known as programming languages.

We shall use meaningful mnemonic variable names, assignment, symbol ← constructions employing **If-then-else, Repeat-until, While-Do** and other constructions employing word **For** for writing an algorithm. We shall also require instructions to input data in an algorithm as well as instructions to output computed results from an algorithm. All these will constitute our language to present any algorithm. This language will not resemble in total with any actual existing programming language but will have desirable characteristics of a good programming language. We shall call it a pseudo-language.

9.1.5 Pseudo language Constructs

(1) **If-then-else construct :** The general form of this construct which is used to provide selection of actions is

```
If condition
   then step
1
```

when an instruction using this type of construct is executed, condition determines which of the step 1 and step 2 is to be executed. If condition is true, step 1 is executed, otherwise (*i.e.,* if condition is not true) step 2 is executed.

A particular case of this construct does not have the word else. The general form of this construct is

```
If condition
   then step
```

Clearly when this form is used, no action is taken when condition is false, and step is executed when condition is true. In other words the following constructs are equivalent as they do the same thing.

If condition **then** step	and	**If** condition **then** step **else** do nothing

(2) **Repeat until construct :** The general form of this construct is

```
Repeat
   Part of the
algorithm
```

This construct is used when repetition of certain action is required. Note that “Part of algorithm” is always executed at least once as the condition is tested at the end, unlike the 'while-do' construct where the condition is tested in the beginning.

(3) **While-do-construct :** This construct is an alternative to the 'repeat-until' construct. The general form of this construct, which is also used to provide repetition of instruction is

```
while
condition
```

Where T is a sequence of instructions. When this construct is executed, condition is evaluated first. If the condition is true, the sequence T of instructions is executed and the condition is evaluated again and so on. If the condition is false, execution of T is skipped and the execution of algorithm proceeds with the portion that appears after T. Thus condition is tested again and again till it is false. Every execution of T modifies some variables in the algorithm and eventually after some repetitions, the condition becomes false. This completes

the execution of the 'while-do-construct', the execution proceeds to the portion appearing after T.

(4) **For construct :** When we know in advance how many times a part of the algorithm is to be executed we use 'for construct', whose general form is

For identifier = initial value to test value by increment **Do** *S*.

The word, **For**, **To, By** and **Do** are reserved words for this construct. Initial value gives the starting value that the identifier should take, when the *S* is executed. The value of identifier is increased by the increment after each execution. The execution of *S* continues until the value of identifier exceeds the test value.

Example: 1 An algorithm must terminate in
(a) One iteration
(b) One step
(c) Finite number of steps
(d) Finite number of steps but sometimes in infinite number of steps

Solution: (c) It is obvious.

Example: 2 The WHILE-DO control structure executes the loop at least
(a) Thrice (b) Twice (c) Once (d) None of these

Solution: (d) It is obvious.

Example: 3 The control structure IF-THEN is a
(a) Multiple selection (b) Double selection (c) Single selection (d) None of these

Solution: (c) It is obvious.

Example: 4 REPEAT-UNTIL control operation executes the loop at least
(a) 3 times (b) 2 times (c) 1 time (d) None of these

Solution: (c) It is obvious.

Example: 5 Write an algorithm to find the first prime number greater than given number using the fact that even integers (except 2) are not prime

Solution:
Step I get n
Step II If n is even
then $m \leftarrow n-1$
else $m \leftarrow n$
Step III repeat $m \leftarrow m+2$
test m for primeness
until m is prime
Step IV Output m

Example: 6 Write an algorithm to find n ! for given n

Solution
Step I get n
Step II If $n < 0$
then output "factorial is not defined"
Step III If $n = 0$
then Fact $\leftarrow 1$
Step IV Fact $\leftarrow 1$
Step V For $I = 1$ to n
do Fact $\leftarrow$ Fact * I
$I \leftarrow I+1$
Step VI Output Fact

Example: 7 Write an algorithm to multiply two matrices

Solution: Step I get A, B

Comment: $A(i, j)$ and $B(i, j)$ are $m \times n$ and $n \times p$ matrices respectively,

Step II
For $i = 1$ to m
do
For $j = 1$ to p
do
$C(i, j) \leftarrow 0$
For $k = 1$ to n
do
$C(i, j) \leftarrow C(i, j) + A(i, k) * B(k, j)$

Step III
Output C

Comment : $C = C(i, j)$ is an $m \times p$ matrix.

Example: 8 Write an algorithm to find the solution of a system of simultaneous linear equations.

$$a_1x + a_2y + a_3z = d_1$$
$$b_1x + b_2y + b_3z = d_2$$
$$c_1x + c_2y + c_3z = d_3$$

Solution:

Step I
get $a(1)$, $a(2)$, $a(3)$, $d(1)$, $b(1)$, $b(2)$, $b(3)$, $d(2)$, $c(1)$, $c(2)$, $c(3)$, $d(3)$, ε

Step II
$I \leftarrow 1, X(1) \leftarrow 0, Y(1) \leftarrow 0, Z(1) \leftarrow 0$

Step III
Repeat

$$X(I+1) \leftarrow \frac{1}{a(1)}[d(1) - a(2)Y(I) - a(3)Z(1)]$$

$$Y(I+1) \leftarrow \frac{1}{b(2)}[d(2) - b(1)X(I \pm 1) - b(3)Z(1)]$$

$$Z(I+1) \leftarrow \frac{1}{c(3)}[d(3) - c(1)$$

$$I \leftarrow I + 1$$

Until
$|X(I) - X(I-1)| < \varepsilon$
$|Y(I) - Y(I-1)| < \varepsilon$
$|Z(I) - Z(I-1)| < \varepsilon$

Step IV
Output $X(I), Y(I), Z(I)$.

9.1.6 Flow Charts (Presentation of Algorithm)

A graphic representation of an algorithm is called a 'flow-chart'. A flow chart constitutes a schematic and pictorial representation of the sequence of steps, which are to be executed in solving a problem.

A flow-chart consists of some boxes linked by arrows. In each box, some instruction to be carried out is mentioned. Arrows on the lines connecting the boxes indicate the direction, in which we should proceed.

The boxes are of different shapes. Each particular shape is associated with a specific type of instruction as shown in fig.

Flow-chart conventions:

While drawing a flow-chart, the following conventions are observed.

(i) The general direction of flow is from left to right and from top to bottom.

(ii) Only one flow-line should leave a process symbol.

(iii) Only one flow line should enter a decision box and atleast two lines must leave it.

(iv) A flow line that goes in upward direction, completes and iteration (or repetition) or a loop.

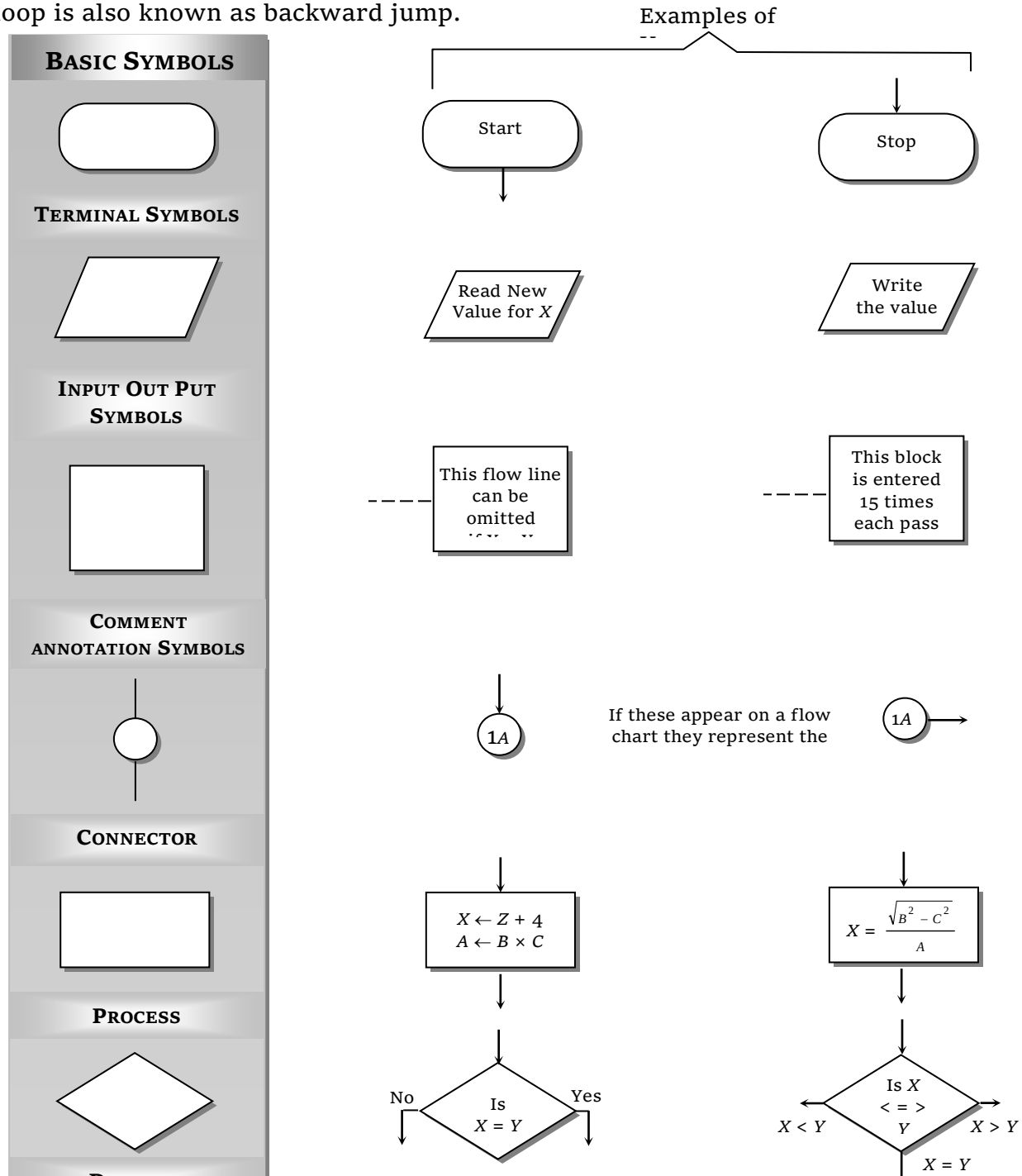
Basic operations and flow-charts:

The three basic operations are: (1) Sequence (2) Selection (3) Iteration

The selection of a flow-chart corresponding to an iteration, or the Repeat-Until construct or the While-Do construct gives rise to a cycle, usually called a loop. There are two types of loops :

(i) When an operation is repeated, a fixed number of times, whatever the value of the variables involved may be, then the corresponding section of the flow diagrams gives rise to a fixed loop.

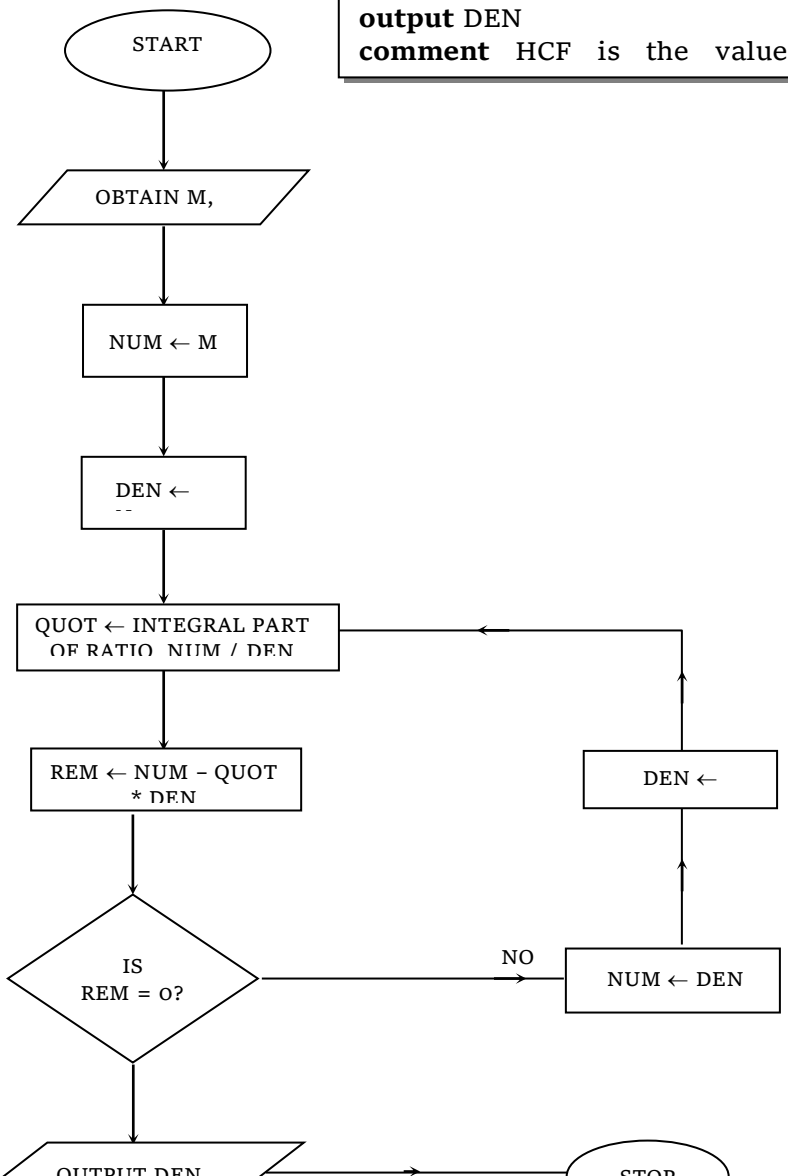
(ii) When the number of times an iteration is to be carried out depends upon the values of the variables, then the corresponding section of the flow diagrams gives rise to variable loop. This loop is also known as backward jump.



Example: 9 Write an algorithm and flowchart to obtain *H C F* of two given positive integers using Euclid's algorithm

Solution: We know Euclid's algorithm. Therefore we can express it in pseudo language :

```
get M, N
comment M, N are two given positive integers, M > N
NUM ← M
DEN ← N
QUOT ← integral part of NUM/DEN
REM ← NUM – QUOT * DEN
while REM ≠ 0
      do NUM ← DEN
         DEN ← REM
          QUOT ← integral part of NUM/DEN
          REM ← NUM – QUOT * DEN
output DEN
comment HCF is the value of denominator when
```



Yes

9.1.7 Number system

(1) **Decimal system :** Number system which we use in our daily life is the decimal system. In decimal system we use the digits namely 0, 1, 2,.....8, 9 and with the help of these 10 digits we are able to write any rational number. The decimal system is a place-value system, meaning thereby that the value represented by a digit depends upon the place of the digit within the numeral. The value assigned to consecutive places in the decimal system are $10^4, 10^3, \ldots\ldots 10^0, 10^{-1}, 10^{-2}\ldots\ldots$ (from left to right)

Example : Number 3864. 342 can be written as

$3864.342 = 3 \times 10^3 + 8 \times 10^2 + 6 \times 10^1 + 4 \times 10^0 + 3 \times 10^{-1} + 4 \times 10^{-2} + 2 \times 10^{-3}$

As ten basic symbols are used for representing the numbers, ten is called the base of the system and the system is called base-ten system or decimal system.

(2) **Binary number system :** The number system for which the base is two is called the binary system. In this system numbers are represented with the help of two basic symbols namely 0 and 1. The values assigned to consecutive places in the system are (when expressed in the decimal system)..... $2^4, 2^3, 2^2, 2^1, 2^0, 2^{-1}, 2^{-2}\ldots$. where 2^0 place is the unit place. The binary numeral can be converted into the decimal numeral and vice-versa.

(3) **Octal number system :** As the name implies this is base eight (2^3) system. The numerals are written with the help of eight basic symbols namely 0, 1, 2,.....,7. The value (expressed in the decimal system) assigned to consecutive places are $\ldots\ldots 8^3, 8^2, 8^1, 8^0, 8^{-1}, 8^{-2}\ldots\ldots$, where 8^0 place is the unit's place. The procedures for converting a decimal numeral into an octal numeral and the other way round are similar to the procedures discussed in connection with binary system.

(4) **Hexadecimal system :** As the name implies, this is the base sixteen system. The numerals are written with the help of sixteen symbols, namely 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Note that the symbol A represents ten (in decimal system). Similarly B, C, D, E, F represent respectively the numbers 11, 12, 13, 14 and 15. The value assigned to consecutive places are ....., $16^2, 16^1, 16^0, 16^{-1}\ldots\ldots$ (as expressed in decimal system), where 16^0 place is unit's place.

Example: 10 74.1875 in binary is **[DCE 2001]**

(a) 10010010.0011 (b) 1001010.0011 (c) 1001010.0101 (d) 1001100.0101

Solution: (b) For integral part

2	74	–
2	37	0
2	18	1
2	9	0
2	4	1
2	2	0
	1	0
		1

$= (1001010)_2$

For fractional part

$$\begin{array}{r} 0.1875 \\ \times \quad 2 \\ \hline \mathbf{0}.3750 \end{array} = (0.0011)_2$$

$$\begin{array}{r} \times \quad 2 \\ \hline \mathbf{0}.7500 \end{array}$$

$$\begin{array}{r} \times \quad 2 \\ \hline \mathbf{1}.5000 \end{array}$$

$$\begin{array}{r} \times \quad 2 \\ \hline \mathbf{1}.0000 \end{array}$$

Therefore 74.1875 in binary is $(1001010.0011)_2$

Example: 11 $(1101101)_2$ is **[DCE 2001]**

(a) 81 (b) 121 (c) 109 (d) 92

Solution: (c) $(1101101)_2 = 1\times2^6+1\times2^5+0\times2^4+1\times2^3+1\times2^2+0\times2^1+1\times2^0$

$= 64 + 32 + 0 + 8 + 4 + 0 + 1 = 109.$

Example: 12 What is the octal equivalent of binary number 111001

(a) 69 (b) 70 (c) 71 (d) 82

Solution: (c) $(111001)_2 =$

$$\underbrace{1\ 1\ 1}_{}\qquad \underbrace{0\ 0\ 1}_{}$$

$$1\times2^2+1\times2^1+2^0 \qquad 0\times2^2+0\times2^1+1\times2^0$$

$$7 \qquad\qquad 1$$

$= (71)_8$

Example: 13 The decimal equivalent of $(264)_8$ is **[DCE 1995]**

(a) 180 (b) 170 (c) 166 (d) None of these

Solution: (a) $(264)_8 = (.....)_{10}$

$= 2\times8^2+6\times8^1+4\times8^0 = 128 + 48 + 4 = 180.$

Example: 14 What is the hexadecimal equivalent of decimal number 785.

(a) 1A2 (b) 311 (c) AB5 (d) None of these

Solution: (b) $(785)_{10} = (.....)_{16}$

16	785	
16	49	1
	3	1
		3

Thus $(785)_{10} = (311)_{16}$.

***

9.2 Binary Operations

9.2.1 Definition

A binary operation on a non-empty set A is a mapping which associates with each ordered pair (a, b) of elements of A, a uniquely defined element $c \in A$. This is a mapping from the product set $A \times A$ to A. Symbolically, a map : $A \times A \to A$, is called a binary operation on the set A.

The image of the element $(a, b) \in A \times A$ is denoted by $a * b$. If a set A is closed with respect to the composition, then we say that * is a binary operation on the set A.

Let, $a \in N, b \in N \Rightarrow a + b \in N$ for all $a, b \in N$.

Multiplication on N is also a binary operation, since $a \in N, b \in N \Rightarrow a \times b \in N$ for all $a, b \in N$

But subtraction on N is not a binary operation, since $3 \in N, 5 \in N$ but $3 - 5 = -2 \notin N$.

Note : ❑ It is obvious that addition as well as multiplication are binary operations on each one of the sets Z (of integer), Q (of rational number), R (of real number) and C (of all complex number).

❑ Subtraction is a binary operation on each of the sets Z, Q, R and C. But it is not binary operation on N.

❑ Division is not a binary operation on any of sets N, Z, Q, R and C.

9.2.2 Types of Binary Operation

(1) **Commutative binary operation :** A binary operation * on a set S is said to be commutative if

$$a * b = b * a \text{ for all } a, b \in S$$

Addition and multiplication are commutative binary operations on Z but the subtraction is not a commutative binary operation, since $2 - 3 \neq 3 - 2$.

(2) **Associative binary operation :** A binary operation * on a set S is said to be associative if

$$(a * b) * c = a * (b * c) \text{ for all } a, b, c \in S$$

Addition and multiplication are associative binary operations on N, Z, Q, R and C. But subtraction is not an associative binary operation on Z, Q, R and C.

(3) **Distributive binary operation :** Let * and o be two binary operations on a set S. Then * is said to be

(i) Left distributive over o if $a * (b \, o \, c) = (a * b) \, o \, (a * c)$ for all $a, b, c \in S$;

(ii) Right distributive over o if $(b\ o\ c) * a = (b * a)\ o\ (c * a)$ for all $a, b, c \in S$.

If * is both left and right distributive over o, then * is said to be distributive over o.

Example **:** The multiplication $(\cdot)$ on Z is distributive over addition (+) on Z, since

$$a \cdot (b + c) = a \cdot b + a \cdot c \text{ and } (b + c) \cdot a = b \cdot a + c \cdot a \text{ for all } a, b, c \in Z.$$

But addition is not distributive over multiplication.

9.2.3 Identity and Inverse elements

(1) **Identity element :** Let * be a binary operation on a set S. An element $e \in S$ is said be an identity element for the binary operation * *if* $a * e = a = e * a$ for all $a \in S$.

For addition on Z, 0 is the identity element, since $a + 0 = a = 0 + a$ for all $a \in Z$.

For multiplication on R, 1 is the identity element, since $1 \times a = a = a \times 1$ for all $a \in R$.

(2) **Inversible element for a binary operation with identity :** An element a of a set A is said to be inversible for a binary operation * with identity e *if* $\exists\ b \in A$ such that $a * b = e = b * a$.

Also, then b is said to be an inverse of a and is denoted by a^{-1}. The inversible elements in A are also called the units in A. The identity element is always inversible and is its own inverse, since $e * e = e * e = e$. Thus $e^{-1} = e$.

9.2.4 Composition Table

A binary operation on a finite set can be completely described by means of a table known as a composition table. Let $S = \{a_1, a_2, ..., a_n\}$ be a finite set and * be a binary operation on S. Then the composition table for * is constructed in the manner indicated below.

We write the elements $a_1, a_2, ... , a_n$ of the set S in the top horizontal row and the left vertical column in the same order. Then we put down the element $a_i * a_j$ at the intersection of the row headed by a_i $(1 \le i \le n)$ and the column headed by $a_j (1 \le j \le n)$ to get the following table.

*	a_1	a_2	.....	a_i	.....	a_j	.....	a_n
a_1	$a_1 * a_1$	$a_1 * a_2$	.....	$a_1 * a_i$	.....	$a_1 * a_j$	.....	$a_1 * a_n$
a_2	$a_2 * a_1$	$a_2 * a_2$	.....	$a_2 * a_i$	.....	$a_2 * a_j$	.....	$a_2 * a_n$
⋮								
a_i	$a_i * a_1$	$a_i * a_2$	.....	$a_i * a_i$	.....	$a_i * a_j$	.....	$a_i * a_n$
⋮								
a_j	$a_j * a_1$	$a_j * a_2$	.....	$a_j * a_i$	.....	$a_j * a_j$	.....	$a_j * a_n$
⋮								
a_n	$a_n * a_1$	$a_n * a_2$	.....	$a_n * a_i$	.....	$a_n * a_j$	.....	$a_n * a_n$

From the composition table we infer the following results :

(1) If all the entries of the table are elements of set S and each element of S appears once and only once in each row and in each column, then the operation is a binary operation. Sometimes we also say that the binary operation is well defined which means that the operation * associates each pair of elements of S to a unique element of S, *i.e.* S is closed under the operation *.

(2) If the entries in the table are symmetric with respect to the diagonal which starts at the upper left corner of the table and terminates at the lower right corner, we say that the binary operation is commutative on S, otherwise it is said to be not commutative on S.

(3) If the row headed by an element say a_j, coincides with the row at the top and the column headed by a_j coincides with the column on extreme left, then a_j is the identity element for the binary operation * on S.

(4) If each row except the topmost row or each column except the left most column contains the identity element then every element of S is invertible with respect to *. To find the inverse of an element say a_j, we consider row (or column) headed by a_i. Then we determine the position of identity element e in this row (or column). If e appears in the column (or row) headed by a_j, then a_i and a_j are inverse of each other.

It should be noted that the composition table is helpless to determine associativity of the binary operation. This has to be verified for each possible trial.

Example: 1 Let S be a finite set containing n elements. Then the total number of binary operations on S is

(a) n^n (b) 2^{n^2} (c) n^{n^2} (d) n^2

Solution: (c) Since a binary operation on S is a function from $S \times S$ to S, therefore the total number of binary operations on S is the total number of functions from $S \times S$ to S, which is n^{n^2}.

Example: 2 The identity element for the binary operation * defined by $a * b = \frac{ab}{2}$, $a, b \in Q_0$ (the set of all non-zero rational numbers) is

(a) 1 (b) 0 (c) 2 (d) None of these

Solution: (c) Let e be the identity element for the binary operation * on Q_0 defined by $a*b = \frac{ab}{2}$

Then, $a*e = a = e*a$ for all $a \in Q_0$

$\Rightarrow \frac{ae}{2} = a$ for all $a \in Q_0 \Rightarrow e = 2$.

Example: 3 Let z be the set of integers and o be a binary operation on z defined as $a\,o\,b = a+b-ab$ for all $a, b \in z$. The inverse of an element $a(\neq 1) \in z$ is

(a) $\frac{a}{a-1}$ (b) $\frac{a}{1-a}$ (c) $\frac{a-1}{a}$ (d) None of these

Solution: (a) Let e be the identity element for the binary operation o defined on z given by $a\,o\,b = a+b-ab$

Then $a\,o\,e = a = e\,o\,a$ for all $a \in z$

$\Rightarrow a+e-ae = a$ for all $a \in z \Rightarrow e(1-a) = 0$ for all $a \in z \Rightarrow e = 0$.

So, 0 is the identity element for the binary operation o and z.

Let x be the inverse of $a \in z$. Then, $a\,o\,x = x\,o\,a = 0$

$\Rightarrow a+x-ax = 0 \Rightarrow x(1-a) = -a \Rightarrow x = \frac{a}{a-1}$ $(\because a \neq 1)$

Thus, $\frac{a}{a-1}$ is the inverse of $a(\neq 1) \in z$.

Example: 4 * is defined on the set of real numbers by $a*b = 1+ab$. Then the operation * is

(a) Commutative but not associative (b) Associative but not commutative

(c) Neither commutative nor associative (d) Both commutative and associative

Solution: (a) We have $a*b = 1+ab = 1+ba = b*a$

So, * is commutative on *R*.

For any, *a, b, c* $\in$ *R*, we have $(a*b)*c = (1+ab)*c = 1+(1+ba)c = 1+c+abc$

and $a*(b*c) = a*(1+bc) = 1+a(1+bc) = 1+a+abc$

$\therefore\ (a*b)*c \neq a*(b*c)$

So, * is not associative on *R*.

***