Name : Priyanka

Roll no. : 4017 / 23

Subject : Computer Oriented Numerical Methods

Submitted to : Ms. Nisha ma’am

UNIT-1
DATA REPRESENTATION AND COMPUTER ARITHMETIC

CONCEPT OF EXACT NUMBER AND APPROXIMATE NUMBER

1. **<u>Exact numbers:</u>** A number is called a exact if it can be represented in finite numbers of digits.

   For example: 3.125 , 7 ,0.45, etc.

2. **<u>Approximate numbers</u>** : A number which can not be expressed by finite number of digits is called approximate numbers.

   For example: π, e, √2, 1.0/3.0 , 2/3 written in this manner are exact whereas if these numbers are written as

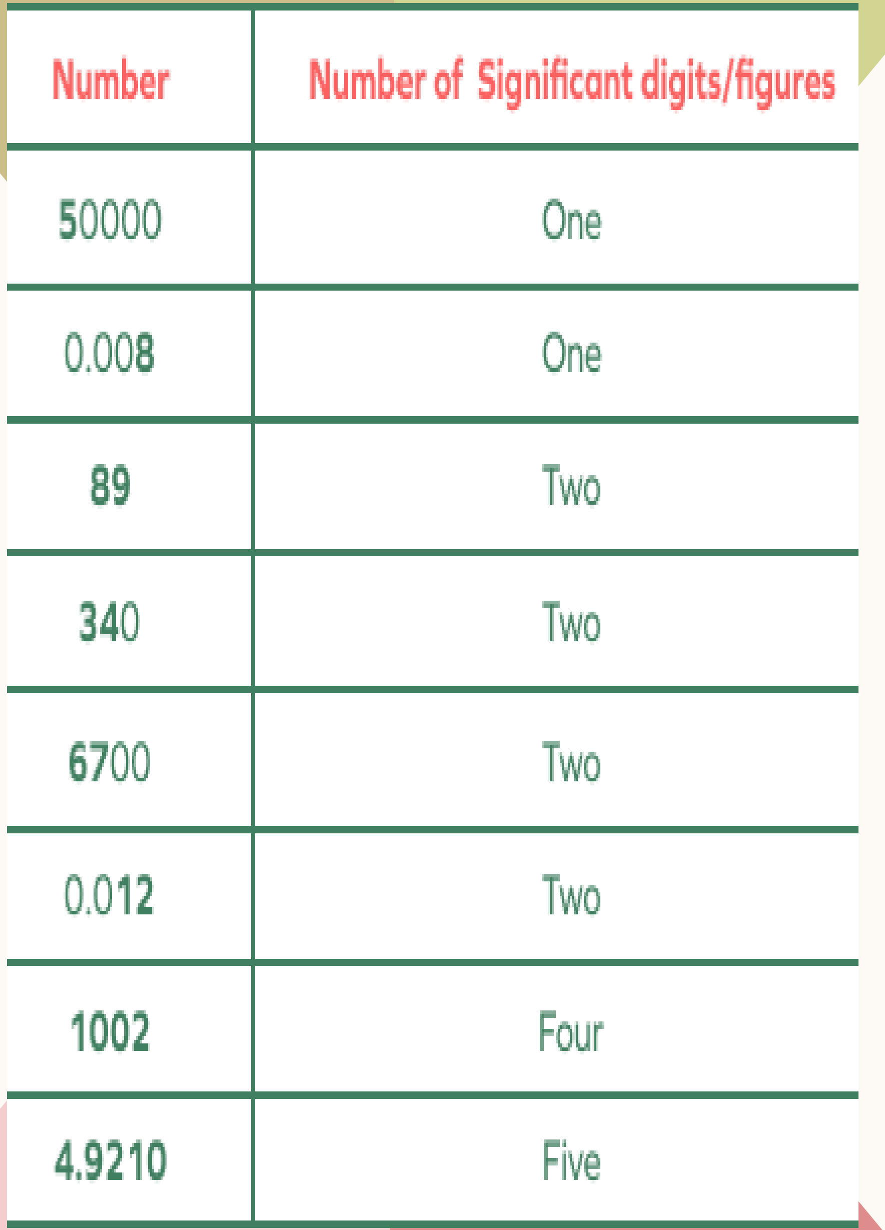
   3.01416 , 2.71828 , 0.333, 0.666 are approximate numbers.

SIGNIFICANT FIGURES

The numbers of digits used to express a given number are called significant digits.

Following rules are observed to describe the significant digits:

1. All non- zero digits are significant.
2. All zeros following a decimal point is significant.
3. Zero between the decimal point are preceding a non-zero digits are not significant.
4. When decimal point is not written, trailing zeros are not considered to be significant.



Number	Number of Significant digits/figures
50000	One
0.008	One
89	Two
340	Two
6700	Two
0.012	Two
1002	Four
4.9210	Five

REPRESENTATION OF NUMBER IN MEMORY

- ➢ Integer number are represented in computer memory, as a sequence of bits.
- • 8 bits = 1 bytes.

- ➢ Therefore in order to store a number, one or more bytes are used.

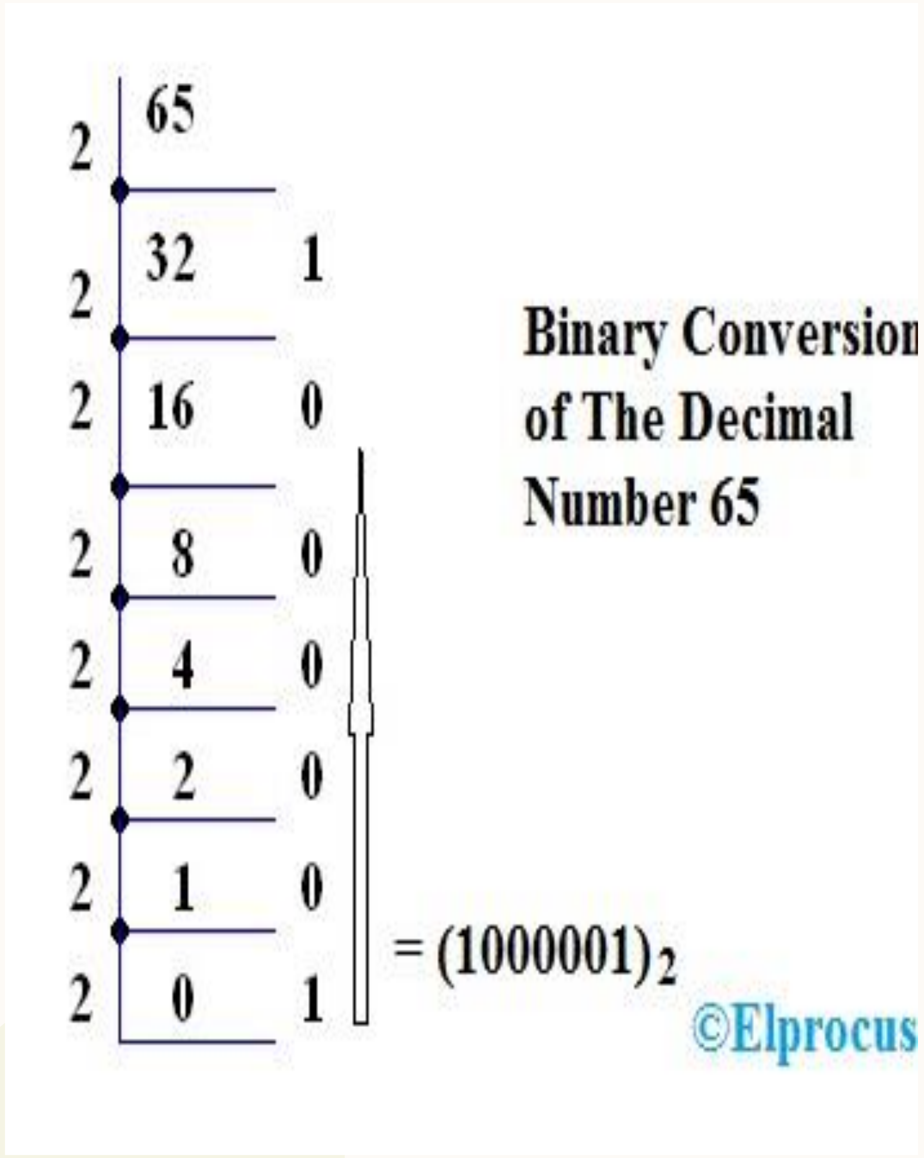
- ➢ Decimal numbers are first converted into their binary equivalent and then are represented in either integer of floating point form.

- ➢ The way of storing a integer number is quite different from that real number.
- ➢ Real number is stored as a floating point number

DECIMAL TO BINARY CONVERSION.

When we enter the data through keyboard, the entered number is converted into binary end and stored in computer's memory.

This shows that the decimal integer number 65=binary integer number 10000001.

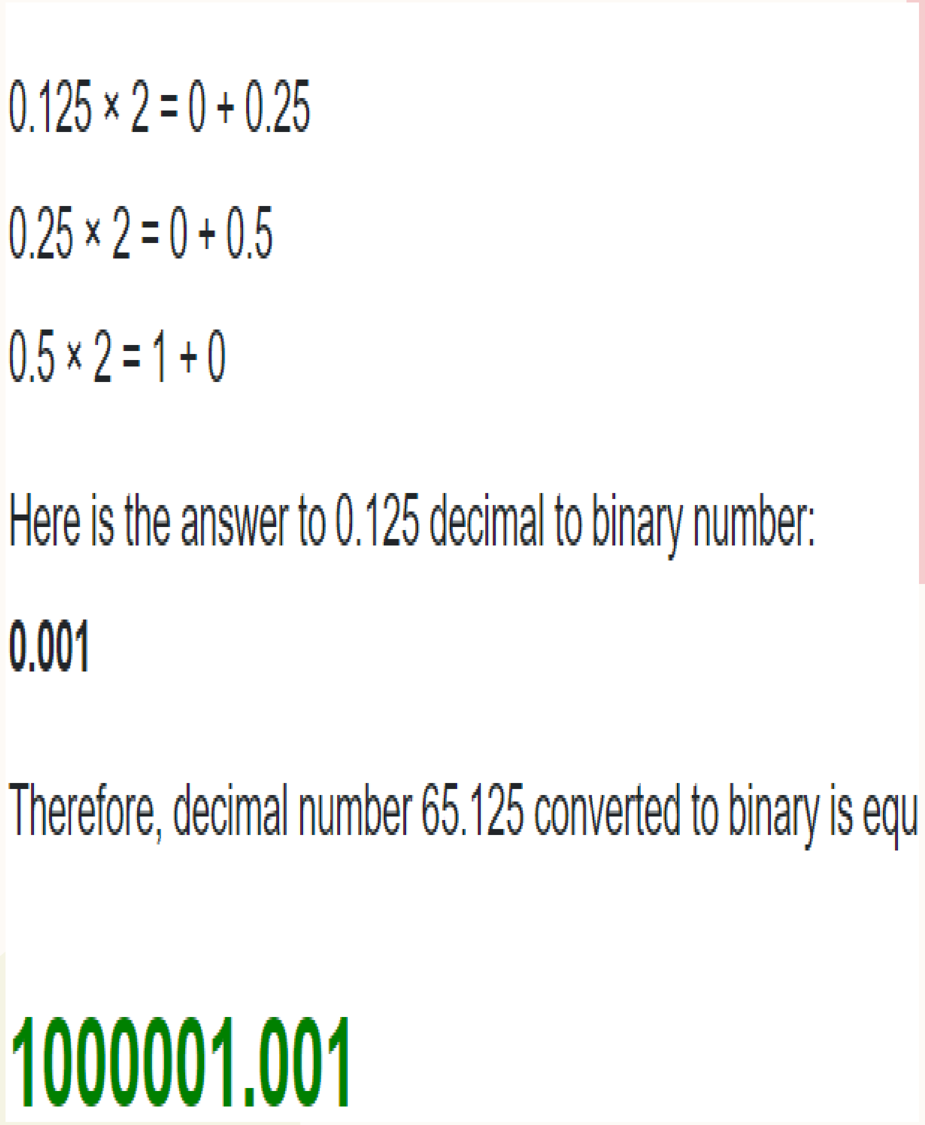


CONVERTING A DECIMAL FRACTION.

The decimal fraction is successively multiplied by 2 till the new fraction is reduced to zero.

Therefore, to covert a real number into binary , the integer part and the fractional part are converted separately, and then combined together to form the equivalent binary.

For example, consider a decimal fraction 65.125



$0.125 \times 2 = 0 + 0.25$

$0.25 \times 2 = 0 + 0.5$

$0.5 \times 2 = 1 + 0$

Here is the answer to 0.125 decimal to binary number:

0.001

Therefore, decimal number 65.125 converted to binary is equ

100001.001

COMPLEMENTS.

In computer, we need to store and represent positive as well as negative numbers. In order to store negative numbers, the complements are used.

We can use 1's complement and 2's complement to represent a negative number.

1's complements:

The 1's complement of binary is obtained by Inverting the bits. i.e., 1 is charged to 0 and 0 to 1.

First we obtain binary representation of number And then we takes its 1's complement , which gives Its final binary representation.

For eg:- decimal 64=1000000

1's complement of 1000000= 0111111

2's complement:

The 2's complement of binary number is taken by first taking 1's complement of it and then adding 1 to the least significant bit (LIB) position of the 1's complement.

For eg:- decimal 64 = binary 1000000

1's complement of 1000000 = 0111111

2's complement of 1000000 =

```
 0111111
      +1
 -------
 1000000
```

STORAGE OF INTEGER NUMBER

Computers have a provision for representing small , large , and very large integer number. These integer are also referred to as 1-byte, 2- byte, and 4-byte integers.
The following are the standard forms for representing the int numbers:

- Signed representation
- 1’s complement representation
- 2’s complement representation

1. Signed representation:
The bits reserved to hold the magnitude of the integer number holds the actual magnitude of the number. For example, the number 44 and -44 in 1-byte format will be represented as 00101100 and 10101100 respectively.

2. 1’s complements representation:
If the sign bit is 0 (positive), the bits reserved to hold the maginitude of integer number holds the actual magnitude of the number.
And if the the sign bit is 1 (negative) , the bits reseverd to hold the maginitude-

Of the integer number hold the 1's complement of the magnitude of the number.
For example, the numbers 44 and -44 in 1-byte format will be represented as 00101100 and 11010011 respectively.

3. 2's complements representation:
if sign bit is 0 (positive) , the bits reserved to hold the maginitude of the integer. And , if the sign bits is 1 (negative), the bits reserved to hold the magnitude of the integer number holds the 2's complement of number
For example the number 44 and -44 in 1-byte format will be represented as 00101100 and 11010100 respectively

Type	Size (in bytes)	Range
Small	1	-128 to +127
Large	2	-32,768 to +32,767
Very large	4	-4,294,967,296 to +4,294,967,295

FLOATING POINT NUMBER AND THEIR STORAGE

Forms of floating point represent :

There are two forms of representation of real number

1. **fixed point representation.:**

- In this form of representation, a real number is written in detail with the decimal point being fixed in correct position between the digits. For example : 125.48 , -3.25 , 0.000789 etc
- This form of representation is convenient, when the number of digits is not very large i.e.,25.35 , 12560.75 etc.
- But if the numbers of digits in a number like 123.000000000573 , 0.00000000 etc is very large the fixed point form of notation is considered inconvenient for that it will consume both space and more time.
- In this form of representation the rational representation like 3/5 , 3/7 etc are not permitted

2. Flaoting point, or Exponential representation form:

- This form of representation of a real number is more convenient, when the value of a number is either very large like 9876543210000000, or very small like.000000001234567.
- A real number is expressed as a combination of mantissa, and exponent by shifting its decimal point by some places either to the right, or to the left according to the requirement, and using the required exponent to the base along with the reduced number.
- The floating point format is similar to scientific-form, where the real number consist of two parts– Mantissa and an exponent. For example, the real number 1.275 x 10^5 in the scientific form will be written as 1.257E5 in the floating-point format.

- **NORMALIZATION:**

The shifting of the decimal point to the left of the most significant digit is called normalization and the real number represented in this form are known as normalized floating-point number.

It consists of two parts – mantissa that is fractional part , and the exponent.

Mantissa of a normalized floating-point number must satisfy

$0.1 <= | \text{mantissa} | < 1.0$.

32-BIT REPRESENTATION

Represent the number 12.25 in 32-bit format.
Step 1: the decimal number 12.25 is converted into binary. The binary equivalent of 12.25 is 01100.01
Step 2 : converting binary number 011000.01 into normalized form we obtain .110001 x 2^4.
Step 3 : the mantissa is 110001, when extended to 23 bits, by adding 0's on the right becomes = 11000100000000000000000
Step 4 : here actual exponent is 4. adding constant 128 to it, the modified exponent becomes 132 . The binary equivalent of 132 is 10000100.
Step 5 : as the number is positive, the sign-bit is 0.
Step 6 : combining the result of all the above steps, we obtain the final representation 12.25 as 10000

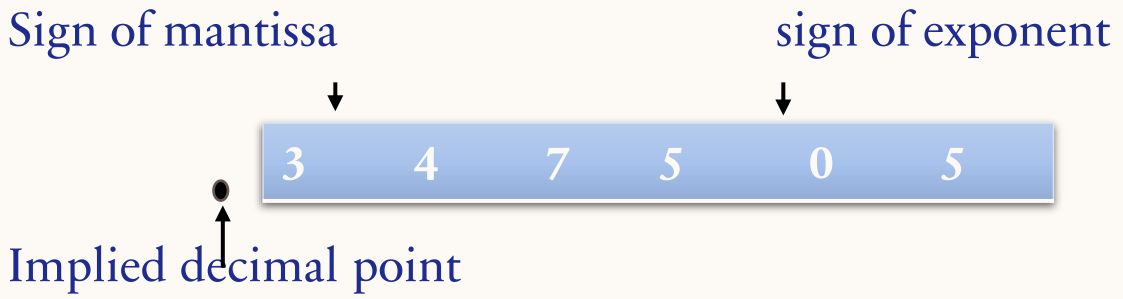
Biased exponent (8 bit)	**Sign (1-bit)**	**Mantissa (23-bits)**
10000100	0	11000100000000000000000

FLOATING POINT ARITHMETIC

we consider a hypothetical computer that can store normalized floating-point number with four digit mantissa and two-digit exponent.

mantissa can vary from -0.9999 to +0.9999 and exponent can var from -99 to +99

For eg:- the number -0.3475E5 will be stored as



1. **Addition operation**

Procedure of addition:

1. Number to be added are equal. If not then increase the exponent of the smaller ones upto the value of the bigger exponent by shifting the mantissa part
2. Mantissa of the number do not consist of more than 4 digits. If so chop the excess digits.

3. Numbers must be normalized.

4. Exponent must be equal.

Example: add 0.4543E3 + 0.4675E7

Raising the exponent of the 1st operand to 7 to make it equal with that of the second operand we get the 1st operand modified as 0.000004543E7 = 0.0000E7.

Now , we add the 2 operands as under :

$$\begin{array}{r} 0.0000E7 \\ +0.4675E7 \\ \hline 0.4675E7 \\ \hline \end{array}$$

2. Subtraction operation:

1. Raise the smaller exponent to the value of the larger exponent by shifting the mantissa of the small exponent appropriately.

1. Chop off the digits of the mantissa in excess of 4 digits to its right
2. Numbers must be normalized.
3. Exponent of the number must be equal.

example : subtract 0.5452E-3 – 0.9432E-4

Raising the exponent of the 2nd number from -4 to -3 to equalize it with that of the 1st number we get 2nd number modified as 0.09432E-3 = 0.0943E-3

Now, we perform the subtraction as under:

$$\begin{array}{r} 0.5452\text{E-}3 \\ -0.0943\text{E-}3 \\ \hline \underline{0.4509\text{E-}3} \end{array}$$

3. Multiplication operation:

1. Multiply first, the mantissa part in the ordinary manner and get the product thereof.
2. Add the exponent of the two given numbers in the ordinary manner.
3. Combine the product of the mantissa, and the sum of the exponent thus obtained, and get the result.
4. Normalize the mantissa of the result, if it is otherwise, and the adjust the exponent appropriatley

Example: find the product of 0.1111E10 X 0.1234E15

Solution: 0.1111 X 0.1234 = 0.1370974 and E10 + E15 = E25.

Combining the above two result, we get = 0.01370974E25. normalizing the above result and chopping off the excess digits in the mantissa over four we get

=0.1370974E24 = 0.1370E24

3. Division operation:

1. Divide first, the mantissa of the dividend by the mantissa of the divisor in the ordinary manner.
2. Subtract the exponent of the divisor from that of the dividend.
3. Combine the quotient and the remainder of the exponent thus obtained, and get the gross result.

Example : find the quotient of 0.9998E1 / 0.1000E-99

Solution : $\frac{0.9998}{0.1000}$ = 9.9980 and E1-E-99=E100

Combining the above two result we have = 9.9980E100

Normalizing the above result we get = 0.9998 E101

(the above result overflows as the exponent > 99 if it is <99 then the result will be underflows).

NORMALIZATION AND THEIR CONSEQUENCES

1. It is well known fact that $4x = x + x + x + x$

   However , when arithmetic is performed using normalized floating point representation, the above equation may not hold true.

2. The associative law of arithmetic does not always hold good in the normalized floating – point representation i.e., $(x + y)-z \neq (x - z) + y$

3. The distributive law of arithmetic does not hold good in the normalized floating point representation i.e., $x(y - z) \neq (xy - xz)$

4. The exact equality of a number to zero, as is found in arithmetic, can never be ensured in the normalized floating point number

ERRORS

An error is defined as the difference between the actual value and the approximate value obtained from numerical computation.

Error = actual value – approximate value = $x\text{-}x_a$

The errors in the computed result can be classified in following categories :

1. **Errors in input data:**

❖ Data errors : these are introduced due to the incorrectness of physical measurement i.e., measurement of distance , length can't be exact always. These are also known as empirical errors.

❖ Truncation errors: it occurs when some digits from the number are discarded. There are mainly two situation when (I) during the representation of the numbers in normalized floating-point form. (II) during the conversion of a number from one system to another.

❖ Round of errors : these are introduced due to rounding off the number to a fixed number of digits.

round of error = exact value – approximate value

2. Computational errors

- the major sources of computational errors are:
  - Normalized floating-point representation.
  - Truncation of infinite series expansion.
  - Inefficient algorithm.

MEASURES OF ACCURACY

It must be remembered that the round of errors are most difficult to estimate and one has to follow some rules to reduce their effects on the final result. However, the truncation errors can be easily estimated and thus can be effectively reduced. In any case, one needs some measures of accuracy of the result.

The commonly used measures are:

1. **Absolute error :** absolute error is magnitude of difference between the actual value and approximate value. It is denoted by E_a
   i.e., $E_a = |x - x_a|$

2. **Relative error :** relative error is the ratio of the absolute to the actual value of a particular result. It is denoted by E_r
   $E_r = |x - x_a| \;/\; x$

3. Percentage error : when the relative error is expressed in terms of percentage, it is called percentage error. It is denoted by Ep
   Ep = relative error x 100

EXAMPLE

Example : consider a measurement of length of plot is 352 metres. But in actual the length of plot is 350 metres. Find the all types of measures of error.

Solution : Actual Value (x) = 350m

Approximate Value (x_a) = 352m

Absolute Error = $| x – x_a |$

= | 350 – 352 |

= 2m

Relative Error = Absolute error / Actual value

= 2 / 350 = 0.0057m

Percentage Error = Relative Error X 100

= 0.57%

THANK YOU ☺

dd