# 4 Number System

## 4.1 INTRODUCTION TO DIGITAL SYSTEM

Digital system is very common nowadays. A digital system is easier to design. It is a combination of devices that handles logical information represented in binary form.

**Digital computer:**

Digital computers follow a sequence of instructions, called a program, that operates on given data. They have the ability to manipulate and represent discrete elements of information.

**Introduction to number system:**

A number system is a mathematical way of representing numbers. The number system is a basis for counting various items.

**Base of number system:**

It is also called as radix of a number system, denoted by 'r'.

The base/radix of any number system is the total number of digits/symbols used in that number system.

It decides the total number of digits available in that system.

For example, for a decimal number system, the base is 10, i.e. (r = 10)

**Types of number systems:**

The commonly used number systems are:

**1)** Decimal number system (r = 10)

**2)** Binary number system (r = 2)

**3)** Octal number system (r = 8)

**4)** Hexadecimal number system (r = 16)

### 4.2 BINARY TO DECIMAL CONVERSION

Using the positional weight method, we can convert binary numbers to their decimal equivalents. In positional weight, each binary digit of the number is multiplied by its positional weights, and to obtain the final decimal numbers, all the product terms are added.

## SOLVED EXAMPLES

**Q1** **Convert $(110010)_2$ to decimal.**

**Sol:** $(1\ 1\ 0\ 0\ 1\ 0\ )_2$

Decimal equivalent:

$= 2^5 \times 1 + 2^4 \times 1 + 2^3 \times 0 + 2^2 \times 0 + 2^1 \times 1 + 2^0 \times 0$

$= 32 + 16 + 0 + 0 + 2 + 0$

$= (50)_{10}$

**Q2** **Convert $(011010.011)_2$ to decimal.**

**Sol:** $(0\ 1\ 1\ 0\ 1\ 0 \cdot 0\ 1\ 1\ )_2$

Decimal equivalent:

$= 2^5 \times 0 + 2^4 \times 1 + 2^3 \times 1 + 2^2 \times 0 + 2^1 \times 1 + 2^0 \times 0 + 2^{-1} \times 0 + 2^{-2} \times 1 + 2^{-3} \times 1$

$= 0 + 16 + 8 + 0 + 2 + 0 + 0 + 0.25 + 0.125$

$= (26{\cdot}375)_{10}$

**Decimal to binary conversion:**

- Decimal to binary conversion is done using successive division and multiplication methods.
- The decimal integer is converted to a binary integer by successively dividing by 2, and a decimal fraction is converted to binary by successively multiplying by '2'.
  Decimal fractions are successively multiplied till we get the fraction part of the product as 0.

## SOLVED EXAMPLES

**Q3** **Convert $(52)_{10}$ to binary.**

**Sol:**

| | | Remainder |
|---|---|---|
| 2 \| 52 | | |
| 2 \| 26 | – | 0 |
| 2 \| 13 | – | 0 |
| 2 \| 6 | – | 1 |
| 2 \| 3 | – | 0 ↑ |
| 1 | – | 1 |

Reading the remainders from bottom to top, the result is $(52)_{10} = (110100)_2$.

**Q4** **Convert $(0.75)_{10}$ to binary.**

**Sol:**

| | |
|---|---|
| **Given fraction** | 0.75 |
| Multiply 0.75 by 2 | 1.50 |
| Multiply 0.50 by 2 ↓ | 1.00 |

Reading the integers from top to bottom, $(0.75)_{10} = (0.11)_2$.

**Q5** **Convert $(105.15)_{10}$ to binary.**

**Sol:** Conversion of integer 105

| | |
|---|---|
| 2 \| 105 | |
| 2 \| 52 | – 1 |
| 2 \| 26 | – 0 |
| 2 \| 13 | – 0 |
| 2 \| 6 | – 1 |
| 2 \| 3 | – 0 |
| 1 | – 1 |

Reading the remainders from bottom to top.

$$(105)_{10} = (1101001)_2$$

Conversion of fraction $(0.15)_{10}$

| | |
|---|---|
| **Given fraction** | → 0.15 |
| Multiply 0.15 by 2 | → 0.30 |
| Multiply 0.30 by 2 | → 0.60 |
| Multiply 0.60 by 2 | → 1.20 |
| Multiply 0.20 by 2 | → 0.40 |
| Multiply 0.40 by 2 | → 0.80 |
| Multiply 0.80 by 2 | → 1.60 |

This fraction can not be expressed in binary exactly. This process may be terminated after a few steps.
Reading the integers from top to bottom.

$$(0.15)_{10} = (0.001001)_2$$

Therefore, the final result is $(105.15)_{10} = (1101001.001001)_2$.

### 4.3 OCTAL NUMBER SYSTEM

It is also a positional-weighted system. Its base is 8. Since Its base $8 = 2^3$. The binary number can be grouped into 3, and each group can be represented by an equivalent octal digit.

**Octal to decimal conversion:**
To convert an octal number to its decimal equivalent, each digit present in the octal number gets multiplied by its positional weight, and then each product term gets added.

## SOLVED EXAMPLES

**Q6** **Convert $(4057.06)_8$ to decimal.**

**Sol:**
$$(4057.06)_8 = 4 \times 8^3 + 0 \times 8^2 + 5 \times 8^1 + 7 \times 8^0 + 0 \times 8^{-1} + 6 \times 8^{-2}$$
$$= 2048 + 0 + 40 + 7 + 0 + 0.0937$$
$$= (2095.0937)_{10}$$

**Decimal to octal conversion:**
To convert a mixed decimal number to a mixed octal number, convert the integer and fraction parts separately.

**Q7** **Convert $(378.93)_{10}$ to octal.**

**Sol:**

| Successive division | | | Remainders |
|---|---|---|---|
| 8 \| 378 | — | | 2 ↑ |
| 8 \| 47 | — | | 7 ↑ |
| 8 \| 5 | — | | 5 ↑ |
| 0 | | | |

Read the remainders from bottom to top.

$$(378)_{10} = (572)_8.$$

Conversion of $(0.93)_{10}$ to octal

$$0.93 \times 8 = 7.44$$
$$0.44 \times 8 = 3.52$$
$$0.52 \times 8 = 4.16$$
$$0.16 \times 8 = 1.28$$

Read the integers to the left of the octal point downwards.

Therefore, $(0.93)_{10} = (0.7341)_8$

Hence, $(378.93)_{10} = (572.7341)_8$.

**Binary to octal conversion:**

A binary number can be converted into octal numbers by grouping the binary number into 3, and each group of 3 can be represented by an octal digit.

## SOLVED EXAMPLES

**Q8** **Convert $(110101.101010)_2$ to octal.**

**Sol:** Groups of three bits 110 101 101 010

Convert each group of 3 bits to octal 6 5 5 2

The result is $(65.52)_8$.

### 4.4 HEXADECIMAL NUMBER SYSTEM

A Hexadecimal number system is a positional-weighted system. The base of this number system is 16, i.e., it has 16 independent symbols. The symbols used are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

**Hexadecimal to decimal conversion:**

To convert a hexadecimal number to its decimal equivalent, each digit present in the hexadecimal number gets multiplied with its positional weight and then each product term gets added.

**Decimal to hexadecimal conversion:**

To convert a decimal integer number to hexadecimal, the decimal number is successively divided by 16 till the quotient is 0.

## SOLVED EXAMPLES

**Q9** **Convert $(2598.675)_{10}$ to hexadecimal.**

**Sol:**

| Successive division | | Remainder | |
|---|---|---|---|
| | | Decimal | Hex |
| 16 \| 2598 | — | 6 | 6 |
| 16 \| 162 | — | 2 | ↑ 2 |
| 16 \| 10 | — | 10 | A |
| 0 | | | |

Reading the remainder upwards, $(2598)_{10} = (A26)_{16}$

Conversion of $(0.675)_{10}$

$0.675 \times 16 \quad — \quad 10.8$ ↓
$0.800 \times 16 \quad — \quad 12.8$
$0.800 \times 16 \quad — \quad 12.8$
$0.800 \times 16 \quad — \quad 12.8$

Reading the integers to the left of hexadecimal point downwards,

$$(0.675)_{10} = (0.ACCC)_{16}$$

Therefore, $(2598.675)_{10} = (A26.ACCC)_{16}$

**Hexadecimal to binary conversion:**
To convert a hexadecimal number to it's equivalent binary representation, each hexadecimal digit is replaced by its 4 bit equivalent binary.

## SOLVED EXAMPLES

**Q10** **Convert $(2BAC)_{16}$ to binary.**

**Sol:** $(2\ B\ A\ C)_2$
Convert the earn digit to its 4-bit binary equivalent

| 2 | B | A | C |
|---|---|---|---|
| 0010 | 1011 | 1010 | 1100 |

**Q11** **Convert $(3A9E.B0D)_{16}$ to binary.**

**Sol:**

| Given hex number is | 3 | A | 9 | E | B | 0 | D |
|---|---|---|---|---|---|---|---|
| Convert each hex digit to 4-bit binary | 0011 | 1010 | 1001 | 1110 | 1011 | 0000 | 1101 |

The result is $(0011\ 1010\ 1001\ 1110\ .\ 1011\ 0000\ 1101)_2$

**Binary to hexadecimal conversions:**
To convert a binary number to its equivalent hexadecimal number,make each group of 4-bits (if there is no decimal point, then start grouping from LSB, and if there is a decimal point, then start grouping from LSB before the point and start grouping from MSB after the decimal point) and replace each group with its equivalent hexadecimal digit.

## SOLVED EXAMPLES

**Q12** **Convert $(001011011011)_2$ to hexadecimal.**

**Sol:**

| Group of 4 bits are | 0010 | 1101 | 1011 |
|---|---|---|---|
| Convert each group to hexadecimal | 2 | D | B |

The result is $(2DB)_{16}$.

## 4.5 BINARY CODED DECIMAL (BCD)

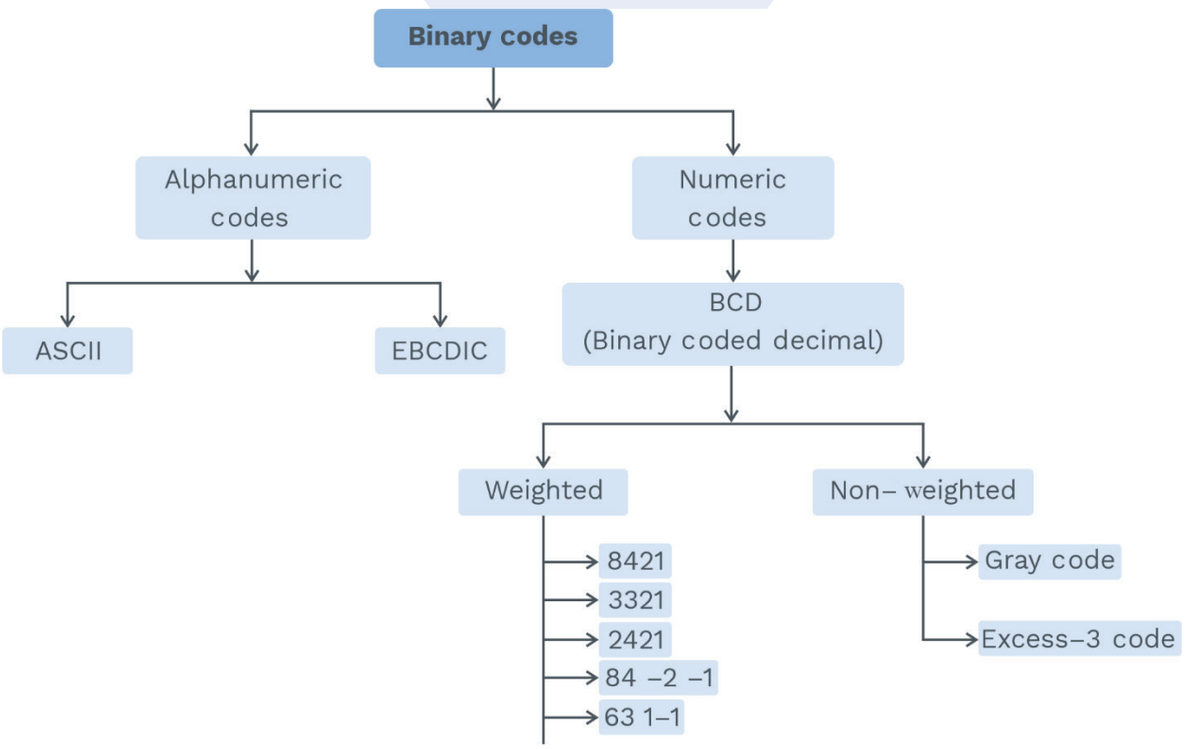
In this type of code, each digit of the decimal number is represented by its 4-bit binary equivalent. It is known as natural binary code due to the positional weights of 8,4,2,1.

**Grey Matter Alert!**

- The decimal 14 can be represented as 1110 in pure binary but as 00010100 in 8421 code.
- Another disadvantage of the BCD code is that arithmetic operations are more complex than they are in pure binary.

In BCD, there exist six combinations that are illegal and are not used 1010, 1011, 1100, 1101, 1110, and 1111, i.e., they are not part of the 8421 BCD code.

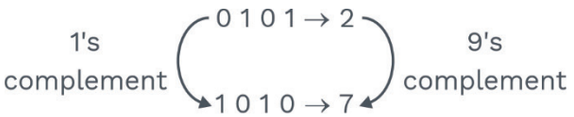
**Classification of binary codes:**




**Self complementary code:**

In self complementary code, the code of a digit and the code of 9's complement of the digit are 1's complement to each other.

**Example:** Let's consider 6 3 1 '-1' code




**Q13** **What is the maximum n-bit number in base x, when represented in decimal (10)?**

**Sol:** A maximum n-bit number possible in base x → In base 'x', maximum digit possible is 'x – 1'. Eg. For x = 10, maximum digit possible is 9.

$$\left(\frac{x-1}{n-1}\cdots\frac{x-1}{2}\,\frac{x-1}{1}\,\frac{x-1}{0}\right)_x$$

↓ in decimal (base 10)

$x^{n-1}(x-1) + x^{n-2}(x-1) + \ldots (x-1)x^0$

$= (x-1)\,[1 + x + x^2 + \ldots x^{n-1}]$

$$= (x-1)\left[\frac{x^n-1}{x-1}\right]$$

$= (x^n - 1)_{10}$

**Rack Your Brain**

What is the minimum number of bits required when $(617)_{10}$, is represented in base 4 number system?

## SOLVED EXAMPLES

**Q14** **What is the minimum number of bits required to represent a 32 bit decimal number in binary number?**

**Sol:** Let's say 'n' bits in binary is required to represent a 32-bit decimal number.

Maximum 32-bit number in decimal = $10^{32} - 1$, and with n-bits, the maximum number in binary when represented in base 10 = $2^n - 1$

Now, $10^{32} - 1 \leq 2^n - 1$

$\Rightarrow$ $10^{32} \leq 2^n$

$\Rightarrow$ $n \geq \log_2 10^{32}$

$\Rightarrow$ $\boxed{n \geq 32 \log_2 10}$

**Q15** **What is the minimum number of digits in base 2 required to represent a 10 digit number in base 8?**

**Sol:** Let's say n bits are required to represent a 10-bit number of base 8 in binary

maximum 10 digit number in base 8 when represented in decimal = $8^{10} - 1$.

With 'n' bits, the maximum number in binary when represented in base 10 = $2^n - 1$

$2^n - 1 \geq 8^{10} - 1$

$\Rightarrow$ $2^n \geq 8^{10}$

$\Rightarrow$ $n \geq \log 8^{10}$

$\Rightarrow$ $n \geq 10 \log 2^3$

$\Rightarrow$ $\boxed{n \geq 30}$

Ans. 30.

**Q16** **Solve for a, b, c**
**$(11)_2 + (22)_3 + (33)_4 + (44)_5 = (abc)_6$**

**Sol:** Let's convert all the number to base 10

$(11)_2 = (3)_{10}$

$(22)_3 = (8)_{10}$

$(33)_4 = (15)_{10}$

$(44)_5 = (24)_{10}$

Now, $3 + 8 + 15 + 24 = (50)_{10}$

Let's convert $(50)_{10}$ to base 6 number.

$$\begin{array}{r|l} 6 & 50 \\ 6 & 8 \\ \hline & 1 \end{array} \begin{array}{l} -2 \\ -2 \end{array}$$

a = 1
b = 2
c = 2

Ans. $(122)_6$

**Q17** **From the following, determine the possible values of x –**
**$(123)_5 = (x8)_y$**

**Sol:** For base 'b' number, digits range from '0' to 'b – 1'

**Example:** for the decimal number system (base 10), digit ranges from '0' to '9'
From the given equation following conditions are derived.

**i)** x < y
**ii)** y > 8

Converting all the number in base 10 is:
$(38)_{10} = xy + 8$

x * y = 30

| x | y | | | |
|---|---|---|---|---|
| 3 | 10 | ✓ | | |
| 5 | 6 | × | Not possible (y > 8) | (Condition not satisfied) |
| 1 | 30 | ✓ | | |
| 2 | 15 | ✓ | | |
| 10 | 3 | × | Not possible (x < y) | (Condition not satisfied) |
| 6 | 5 | × | Not possible (x < y) | |
| 30 | 1 | × | Not possible (x < y) | |
| 15 | 2 | × | Not possible (x < y) | |

Ans.3, 1, 2

**Q18** **Determine the base of the system.**
**Roots of $x^2 - 11x + 22 = 0$ are '3' and '6'.**

**Sol:** Product of roots of $ax^2 + bx + c = 0$ is $\frac{c}{a}$

Let 'b' be the base of the system

$$(3)_b \times (6)_b = \frac{(22)_b}{(1)_b}$$

$$\Rightarrow \quad (3)_{10} \times (6)_{10} = \frac{2 \times b + 2}{(1)_{10}}$$

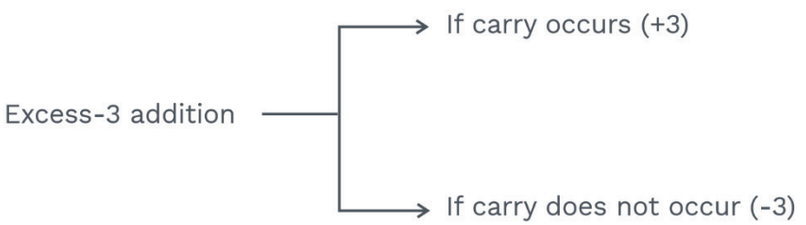
$$\Rightarrow \quad 18 = 2b + 2$$

$$\Rightarrow \quad 2b = 16$$

$$\boxed{b = 8}$$

**Excess-3 code:**
It is a non-weighted BCD code. Each excess-3 codeword is the corresponding 8-4-2-1 code word push 3 (0011). It is a self complementing code. It is a sequential code.
The excess-3 code has six invalid states 0000, 0001, 0010, 1101, 1110, and 1111.

**Excess-3 addition:**
The addition of two numbers in excess-3 is performed in a group of 4-bits starting from LSB and adding them column-wise. If there is a carry out from a group, then add 0011 to the sum term of the group and if there is no carry out then subtract 0011 from the sum term of that group.



## SOLVED EXAMPLES

**Q19** **Perform Excess-3 addition of i) and ii)**
**i)** 7 + 2 = ?  **ii)** 7 + 3 = ?

**Sol:** **i)**

$7 \rightarrow 1010$
$+2 \rightarrow 0101$
$9 \rightarrow 1111 \longrightarrow$ No carry
$-\ 0011$
$(1100)$ $\rightarrow$This is 9 in excess - 3

**ii)**

carry
1

| | | | |
|---|---|---|---|
| 07 | 0011 | 1010 | - '07' in excess-3 |
| +03 | +0011 | 0110 | - '03' in excess-3 |
| 10 | 0111 | 0000 | |
| | -0011 | +0011 | |
| | 0100 | 0011 | |

0100 ← 1 in excess – 3
0011 ← 0 in excess – 3

**Q20** **Perform the following addition in excess – 3 code.**
**a) 37 + 28 = ?**  **b) 247.6 + 359.4**

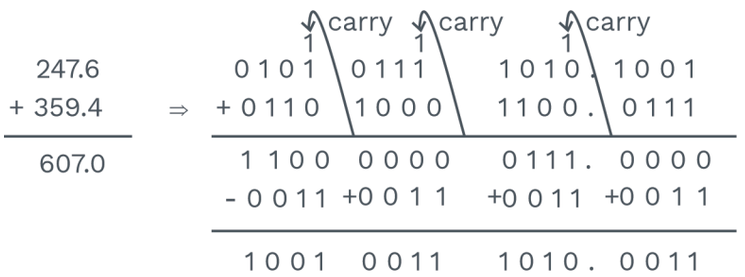
**Sol:** **a)**

carry
1

| | | | |
|---|---|---|---|
| 37 | 0110 | 1010 | (37 in Excess - 3) |
| +28 | +0101 | 1011 | (28 in Excess - 3) |
| 65 | 1100 | 0101 | |
| | -0011 | +0011 | |
| | 1001 | 1000 | |

1 0 0 1 ← '6' in excess – 3
1 0 0 0 ← '5' in excess – 3

**b)**




## 4.6 GRAY CODE

This type of code is also known as cyclic code/unit distance code/reflective code, as any two succeeding binary number equivalent gray codes differ in only one bit. It is a non-weighted code and is not well suited for arithmetic operations.

**Grey Matter Alert!**

Gray code is reflective. The n least significant bits for $2^n$ through $2^{n+1} - 1$ are the mirror images of those for 0 through $2^n - 1$. An N-bit gray code can be obtained by reflecting an N-1 bit code about an axis at the end of code and putting the MSB of 0 above the axis and MSB of 1 below axis.

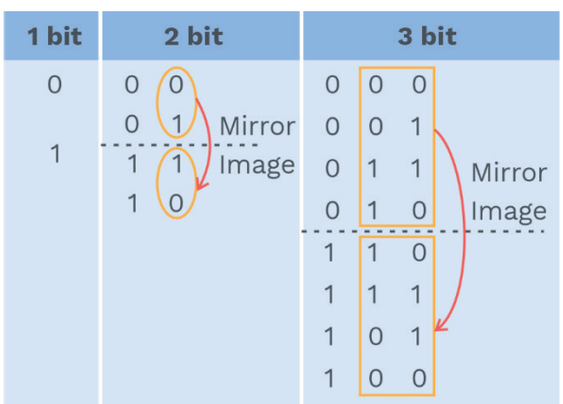
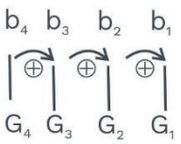



**Table 4.1 Table for Gray Code**

**Binary to gray conversion:**
Let the binary number be $b_4\, b_3\, b_2\, b_1$

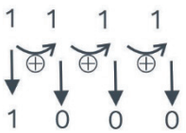



$$G_4 = b_4$$
$$G_3 = b_4 \oplus b_3$$
$$G_2 = b_3 \oplus b_2$$
$$G_1 = b_2 \oplus b_1$$

## SOLVED EXAMPLES

**Q21** **Convert $(1111)_2$ to gray code.**

**Sol:**

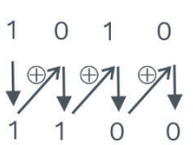



1000 in grey – Ans.

**Grey to binary code:**
Let the binary number be $b_4\, b_3\, b_2\, b_1$

$$b_4 = G_4$$
$$b_3 = b_4 \oplus G_3 = G_4 \oplus G_3$$
$$b_2 = b_3 \oplus G_2 = G_4 \oplus G_3 \oplus G_2$$
$$b_1 = b_2 \oplus G_1 = G_4 \oplus G_3 \oplus G_2 \oplus G_1$$

**Example:**




$(1\,1\,0\,0)_2$

## 4.7 COMPLEMENTARY NUMBER SYSTEM

Advantage → Subtraction can be performed using addition

Base b
- → (b-1)'s complement - diminished radix complement
- → b's complement - radix complement

⇒ Diminished radix complement of 'x' in base 'b':

$$\boxed{(b^n - 1) - x}$$

⇒ Radix complement of 'x' in base 'b':

$$\boxed{b^n - x}$$

**Example:**

Base 2
- → 1's complement $\boxed{\overline{x} = (2^n\text{-}1) - x}$
- → 2's complement $\boxed{\overline{x} = 2^n - x}$

Base 3
- → 2's complement $\boxed{\overline{x} = (3^n\text{-}1) - x}$
- → 3's complement $\boxed{\overline{x} = 3^n - x}$

Base 2 1's complement → $(2^n - 1) - x$

$x = 1011$

$\overline{x} = (2^4\text{-}1)\text{-}1011$

$\overline{x} = 1111$

$\quad - 1011$

$\quad 0100$

2's complement = 1's complement + 1

$= (2^n - 1) - x + 1$

$= 2^n - x$

$\quad 0100$

$+ \quad 1$

$\quad 0101$

Base – 16 15's complement → $(16^n - 1) - x$

$16^{th}$ complement → $16^n - x$

$x = 1234$

$\overline{x}$ (in 15's complement) = $\begin{array}{r} FFFF \\ 1234 \\ \hline (EDCB)_{16} \end{array}$

$\overline{x}$ (in 16's complement) → $\begin{array}{r} EDCB \\ + \quad 1 \\ \hline (EDCC)_{16} \end{array}$

## SOLVED EXAMPLES

**Q22** **Find 9's complement of $(685)_{10}$.**

**Sol:** $(10^n - 1) - 685$

here n = 3

$(10^3 - 1) - 685 = 999 - 685 = 314$

** To find 10's complement = 9's complement + 1

= 314 + 1 = 315

**Q23** **Perform the subtraction of the following numbers of base 10 using 9's complement:**
**a) 97-23** **b) 23-97**

**Sol:** **a)** For the general case,

$$
\begin{aligned}
& x-y \\
&= x+(\bar{y}) \\
&= x+(10^2-1-y) \\
&= x-y+99 \\
&= (x-y)+99-100 \\
&= (x-y)-1 \\
&= (x-y)-1+1 \\
&= x-y
\end{aligned}
$$

→ carry exists [Discard carry, Add +1]
→ carry does not exist

$$97 - 23 = 97 + (-23)$$

9's complement of 23 →

$$\begin{array}{r} 99 \\ -23 \\ \hline 76 \end{array}$$

$$\begin{array}{r} 97 \\ +\ 76 \\ \hline (1)\,73 \end{array}$$

Discarded

$$\begin{array}{r} 73 \\ +1 \\ \hline 74 \end{array} \rightarrow \text{Ans}$$

**b)** 23 – 97

9's complement of 97 →

$$\begin{array}{r} 99 \\ -97 \\ \hline 02 \end{array}$$

23 + (-97)
= 23 + 02
= 25 (No carry)
↓ Taking 9's complement

99
- 25
74
→ (-74) Ans
Negating it

**Q24** **Perform the following subtraction using 1's complement for the following binary numbers.**
**a) 1011 – 0101 = ?** **b) 111 – 10100**

**Sol:** **a)**

$1011 \leftarrow (11)_{10}$
$0101 \leftarrow (5)_{10}$
↓ Taking 1's complement
1010

1010
+ 1011
(1) 0101
+ 1
0110 - Ans
End around carry is discarded

**b)** x = 111
y = 10100
x–y = ?

$\bar{y} = 01011$

$x+\overline{y} = 00111 + 01011$

$= 10010$ (No carry)

↓ Taking 1's complement

$= 01101$

$= (13)_{10}$

Negating → –13 (Ans.)

**Q25** **Perform the subtraction using b-1's complement for the following numbers in base 3.**
**212 – 121 = ?**

**Sol:** x = 212
y = 121
x–y = ?

$\overline{y} = 222 - 121 = 101$

$x+\overline{y} = 212$
$+\ 101$
$1\,020$
$+\ \ 1$
$(021)_3$ Ans.

End around carry is discarded

** in decimal = $(021)_3 = 3 \times 2 + 1 = (7)_{10}$

**Q26** **Perform the following subtraction using b's complement for the numbers given in binary 0101 – 1011 = ?**

**Sol:** x = 0101
y = 1011
x-y = ?

$y = 1011$

↓ 1's complement

$\overline{y} = 0100$

↓ 2's complement

$\overline{y} = 0100+1$

$= 0101$

$x+(\overline{y}) = 0101$

$+\ 0101$

$1010 \rightarrow$ No carry

↓ 2's complement

$0110$

$-(x-y) = 0110 = (6)_{10}$

$x - y = -6$ Ans.

**Q27** **Perform the following subtraction using b's complement for the given numbers in base 3.**
**$(212)_3 - (121)_3 = ?$**

**Sol:** $x = (212)_3$

$y = (121)_3$

$\overline{y} = 222$

$-\ 121$

$101 \leftarrow$ 2's complement

$+\ 1$

$102 \leftarrow$ 3's complement

$x+\overline{y} = 212 \rightarrow x$

$+\ 102 \rightarrow (b^n-y)$

$(1)021 \rightarrow (x-y)+b^n$

Discarded

Ans $(021)_3$

**Grey Matter Alert!**

**Complementary number system – summary**

The subtraction of two n-bit unsigned number

M-N ($N \neq 0$) in base "b" using diminished radix complement can be done as follows

**1)** Add M to (b-1)'s complement of N. this performs

$= M + (b^n - 1 - N)$

$= (M - N) + b^n - 1$

**2)** If $(M-N) \geq 1$, then sum will produce an end carry $b^n$, will be discarded and 1 is added which gives (M-N)

**3)** if $(M-N) < 1$, then sum does not produce an end carry. To obtain the answer in familiar form, take (b-1)'s complement of the sum and place a negative sign in front.

**Grey Matter Alert!**

**Complementary number system – summary**

The subtraction of two n-bit unsigned number

M-N ($N \neq 0$) in base "b" using radix complement can be done as follows

**1)** Add M to (b)'s complement of N. this performs

$=M + (b-N)$

$= (M-N) + b$

**2)** If $(M-N) \geq 0$, then sum will produce an end carry $b^n$. will be discarded which gives (M-N)

**3)** If $(M-N) < 0$, then sum does not produce an end carry. To obtain the answer in familiar form, take b's complement of the sum and place a negative sign in front.

$b^n - (b^n + (M - N)) - (-(M - N)) = (M - N)$

**Note:**

Let's take b = 2

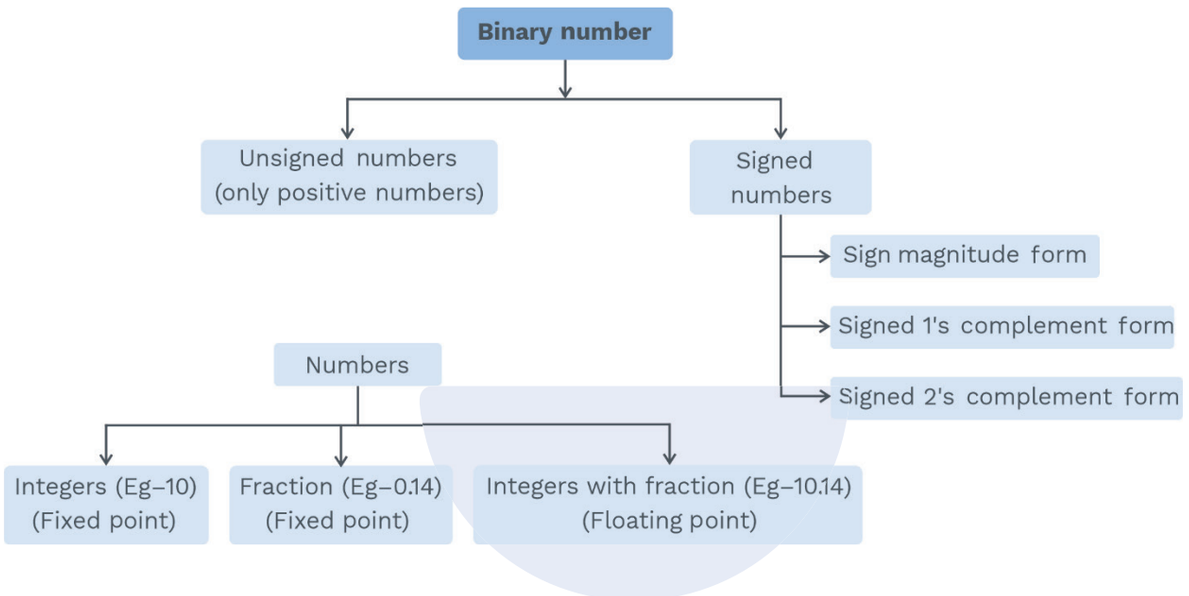
To perform M - N using radix complement we follow the steps as shown below

Step 1: Add M to 2's complement of N.

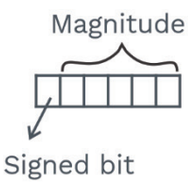
Step 2: If the sum produces carry then it is simply discarded.

Step 3: If the sum does not produce carry then take 2's complement of the sum and place a negative sign in front.

## 4.8 SIGNED NUMBER REPRESENTATION




**Signed magnitude form:**
If MSB is 1, the number is negative.
If MSB is 0, the number is positive.




**Disadvantage:**

**i)** Two combinations for '0' (zero)
**ii)** Signed bit (MSB) needs to be checked to take the decision for addition subtraction.
**iii)** Additional hardware required.

**Signed 1's complement form:**
Understanding through examples:

**Example:**

$(010)_2 = 2$

↓ 1's complement

$(101)_2 = -2$

**Example:**

$(001) = 1$

↓ 1's complement

$(110) = -1$

**Disadvantage**

**Grey Matter Alert!**

**Q.** Let's try to perform x-x using 1's complement (x' is in base 2)

**Sol.**

$X+\overline{X}$

$= X+(2^n-1-x)$

$= (x-x) + (2^n-1)$

$= 0 + \underbrace{(2^n-1)}$

Undesirable term

↓ Solution

End Around Carry is discarded if any.

**2's complement form:**

**Example:**

001 → 1

↓ 1's complement

110

+ 1

111 → -1 (in 2's complement)

**Example:**

010 → 2

↓ 1's complement

101

+ 1

110 → -2 (in 2's complement)

**Weighted code:**

→ Unsigned number is weighted representation.

→ Signed magnitude form is weighted code.

→ 2's complement is weighted code.

**E.g:** 1 0 0

$-2^2\ 2^1\ 2^0$

$= 1 \times (-2^2)$

$= -4$

→ 1's complement is non-weighted.

Examples of signed number representation:

| | Unsigned | SM | 1's | 2's |
|---|---|---|---|---|
| 0 0 0 | 0 | 0 | 0 | 0 |
| 0 0 1 | 1 | 1 | 1 | 1 |
| 0 1 0 | 2 | 2 | 2 | 2 |
| 0 1 1 | 3 | 3 | 3 | 3 |
| 1 0 0 | 4 | 0 | -3 | -4 |
| 1 0 1 | 5 | -1 | -2 | -3 |
| 1 1 0 | 6 | -2 | -1 | -2 |
| 1 1 1 | 7 | -3 | 0 | -1 |

**Table 4.2**

**Ranges:**

| Number of bits | Signed magnitude | 1's | 2's |
|---|---|---|---|
| 3 | (0 to 3) & (0 to -3)<br>OR<br>(-3) to (+3) | (-3) to (+3) | (-4) to (+3) |
| 4 | (0 to 7) & (0 to –7)<br>OR<br>-7 to 7 | (-7) to (+7) | (-8) to (+7) |

**Table 4.3**

**Grey Matter Alert!**

In signed magnitude form, 1's complement form, there are two representation for 0(zero)

**Generalisation for "n" bit numbers:**

Signed Magnitude: $2^n$ → +ve No. → $\frac{2^n}{2} = 2^{n-1}\,[0-(2^{n-1}-1)]$

$2^n$ → -ve No. → $\frac{2^n}{2} = 2^{n-1}\,[-(2^{n-1}-1)-0]$

$$\boxed{-\left(2^{n-1}-1\right) \text{ to } \left(2^{n-1}-1\right)}$$

**2's complement form**

$$\left(0 \text{ to } \left(2^{n-1}-1\right)\right) \& \left(-1 \text{ to } \left(-2^{n-1}\right)\right)$$

$$\boxed{-2^{n-1} \text{ to } \left(2^{n-1}-1\right)}$$

## SOLVED EXAMPLES

**Q28** **To represent 15 in 2's complement number system. What is the minimum number of bits required?**

**Sol:** $\left(2^{n-1}-1\right) \geq 15$

$\Rightarrow 2^{n-1} \geq 16$

$\Rightarrow n-1 \geq 4$

$\Rightarrow n \geq 5$

$n = 5$ Ans.

**What is overflow? How does it happen?**

$\Rightarrow$ In 2's complement, 4-bit range (–8 to 7)

"–16" cannot be represented using 4 bits in 2's complement representation. (overflow)

**Why is overflow condition crucial in computers?**

⇒ While doing arithmetic, the computer stores the result in a register.

If the register is of size 8 bit, the result should not exceed (–128 to 127).

**Q29** **–30 is to be represented in 2's complement number representation. What is number of bits (minimum) required?**

**Sol:** $-2^{n-1} \leq -30$

$2^{n-1} \geq 30$

$n - 1 \geq 4.5$

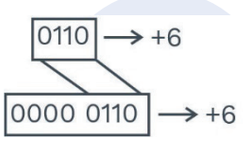
$\boxed{n \geq 5.5}$ Minimum number of bits = 6

**Sign bit extension**

Sign bit extension expands a smaller bit number without changing Its actual value.

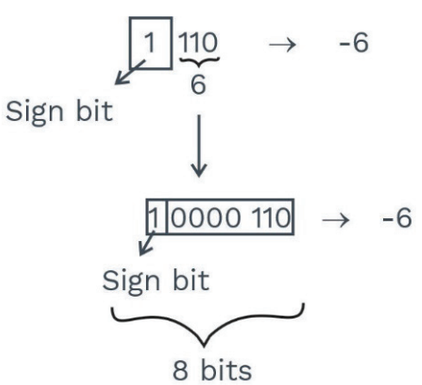
**1) Unsigned number**

Suppose we want to represent 0110 in 8 bits.

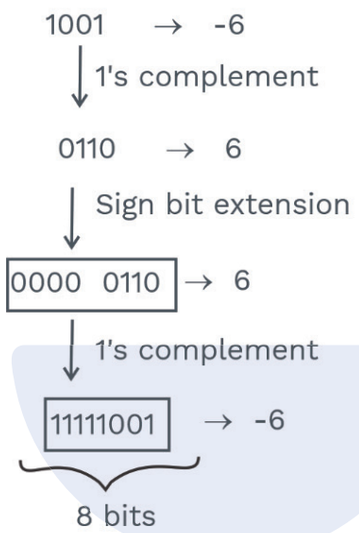



**2) Sign magnitude**

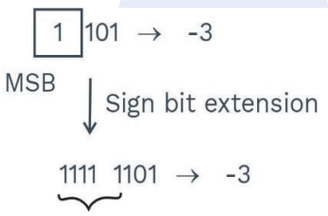
The first bit (MSB) denotes the sign



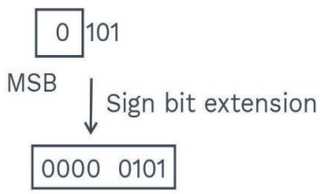
**3) 1's Complement**
**Example:**




**4) 2's Complement**
**Example:**




**Example:**



**Rack Your Brain**

A number representation follows 2's complement representation form. What will be decimal equivalent (base 10) of $(FFF3)_{16}$?

**Previous Years' Question**

The smallest integer that can be represented by an 8-bit number in 2's complement form is **(GATE-2013)**

**1)** –256
**2)** –128
**3)** –127
**4)** 0

**Sol: 2)**

**Previous Years' Question**

P is a 16 bit signed integer. The 2's complement representation of P is $(F\ 87\ B)_{16}$. The 2's Complement representation of 8 × P is **(GATE-2010)**

**1)** $(C3D8)_{16}$
**2)** $(187B)_{16}$
**3)** $(F878)_{16}$
**4)** $(987B)_{16}$

**Sol: 1)**

**Previous Years' Question**

Let r denote number system radix. The only value(s) of r that satisfy the $\sqrt{121_r} = 11_r$ is/are: **(GATE-2008)**

**1)** decimal 10
**2)** decimal 11
**3)** decimal 10 and 11
**4)** any value > 2

**Sol: 4)**

**Rack Your Brain**

A 2's complement number N = $P_3\ P_2\ P_1\ P_0$ is transformed as $P_3\ P_3\ P_3\ P_2\ P_1\ P_0\ 1$. Which of the following operation is performed on 'N'?

**1)** 2N+1
**2)** 2N
**3)** N + 1
**4)** 3N + 1

**Overflow:**

**Definitions**

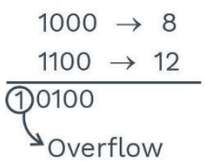
When two numbers of n digits each are added and the sum occupies n + 1 digits, we say that an overflow occurs. (OR) Overflow is said to occur when magnitude of a number exceeds the range allowed by the size of bit-field.

⇒ Addition of two 4-bit numbers might result in 5 bit

$$\begin{array}{r} 1000 \\ +\ 1100 \\ \hline 10100 \end{array}$$

Let's discuss about overflow for different classifications of numbers:

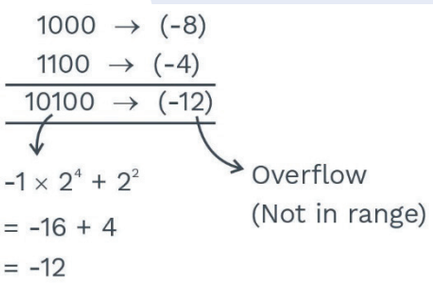
**1) Unsigned number:**



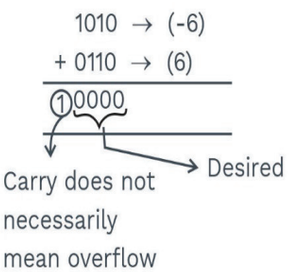

For 4 bits number, number ranges $\left(0-\left(2^4-1\right)\right)$

**2) Signed 2's complement:**

For 4 bit number, range → (−8 to 7)




Let's look at another example:




Let's combine whatever we learned about the Overflow concept till now:

**1)** Overflow condition for unsigned numbers
- Carry always indicates overflow

**2)** Overflow condition for signed numbers.

**Case 1** When two positive numbers are added, the result is negative.

**Example:**

$$\begin{array}{l} 0111 \rightarrow 7 \\ 0001 \rightarrow 1 \\ \hline 1000 \rightarrow 8 \end{array}$$

-ve (Overflow)

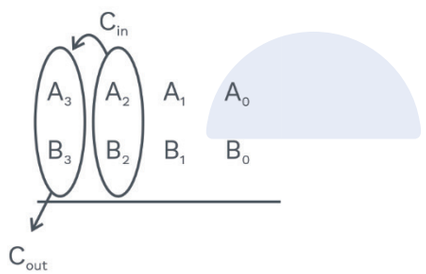
**Case 2** When 2 negative numbers are added and the result is +ve.

**Example:**

$$\begin{array}{l} 1000 \rightarrow (-8) \\ 1100 \rightarrow (-4) \\ + \\ \hline 10100 \\ \hline \end{array}$$

+ve (Overflow)

SHORTCUT (For signed numbers)




$C_{out} \neq C_{in}$ (overflow)
$C_{out} = C_{in}$ (no overflow)


**Rack Your Brain**

In a Number system, range of numbers are –3 to 3 represented as C, B, A, 0, 1, 2, 3 What will be $(102)_{10}$ in this system?

**Previous Years' Question**

When two 8-bit numbers $A_7$ .... $A_0$ and $B_7$ .... $B_0$ in 2's complement representation (with $A_0$ and $B_0$ as the least significant bits) are added using ripple carry adder. The sum bits obtained are $S_7$ .... $S_0$ and the carry bits are $C_7$ .... $C_0$ An overflow is said to have occurred if **(GATE-2008)**

**1)** The carry bit $C_7$ is 1

**2)** All the carry bits ($C_7$ .... $C_0$) are 1

**3)** ($A_7B_7S_7 + A_7'B_7;S_7$) is 1

**4)** ($A_0.B_0.S_0' + A_0;B_0;S_0$) is 1

**Sol: 3)**

### 4.9 ARITHMETIC OPERATIONS

**1) Binary addition:**

The rules for binary addition are the following:

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 10 i.e. 0 with a carry of 1

## SOLVED EXAMPLES

**Q30** **Add the binary numbers 1101.101 and 111.011**

**Sol:**

| $8\ 4\ 2\ 1$ | $2^{-1}2^{-2}2^{-3}$ |
|---|---|
| $1\ 1\ 0\ 1$ . | $1\ 0\ 1$ |
| $1\ 1\ 1$ . | $0\ 1\ 1$ |
| + | |
| $1\ 0\ 1\ 0\ 1$ . | $0\ 0\ 0$ |

In the $2^{-3}$'s column, 1+1 = 0 with a carry of 1 to the $2^{-2}$ column
In the $2^{-2}$'s column, 0+1+1 = 0 with a carry of 1 to the $2^{-1}$ column
In the $2^{-1}$'s column, 1+0+1 = 0 with a carry of 1 to the 1's column
In the 1's column, 1+1+1 = 1 with a carry of 1 to the 2's column
In the 2's column, 0+1+1 = 0 with a carry of 1 to the 4's column
In the 4's column, 1+1+1 = 1 with a carry of 1 to the 8's column
In the 8's column, 1+1 = 0 with a carry of 1 to the 16's column

**2) Binary subtraction:**

Binary subtraction is similar to decimal subtraction and the rules for it are given below:

**i)** 0–0 = 0
**ii)** 1–1 = 0
**iii)** 1–0 = 1
**iv)** 0–1 = 1 with a borrow of 1.

## SOLVED EXAMPLES

**Q31** **Subtract $(111.111)_2$ from $(1010.01)_2$**

**Sol:**

$$\begin{array}{r} 8\,4\,2\,1 \quad 2^{-1}2^{-2}2^{-3} \\ 1\,0\,1\,0 \;.\; 0\,1\,0 \\ \quad 1\,1\,1 \;.\; 1\,1\,1 \\ + \\ \hline 0\,0\,1\,0 \;.\; 0\,1\,1 \end{array}$$

In the $2^{-3}$ column, a 1 can not be subtracted from a 0. So, borrow a 1 from the $2^{-2}$ column making the $2^{-2}$ column 0. The 1 borrowed from the $2^{-2}$ column becomes 10 in the $2^{-3}$ column. Therefore, in the $2^{-3}$ column, 10–1 = 1
In the $2^{-2}$ column, a 1 can not be subtracted from a 0, borrow a 1 from the $2^{-1}$ column, but it is also a 0. So borrow a 1 from the 1's column. That is also a 0, so borrow a 1 from the 2's column making the 2's column 0.

This 1 borrowed from the 2's column becomes 10 in the 1's column. Keep one 1 in the 1's column, bring the other 1 to the $2^{-1}$ column, which becomes 10 in this column.

Keep one 1 in the $2^{-1}$ column and bring the other 1 to the $2^{-2}$ column, which becomes 10 in this column. Therefore,

In the $2^{-2}$'s column 10–1 = 1
In the $2^{-1}$'s column 1–1 = 0
In the 1's column 1–1 = 0

Now, in the 2's column, a'1' cannot be subtracted from a 0, so borrow a 1 from the 4's column. But 4's column has a 0, so borrow a 1 from 8's column, making 8's column 0 and bring it to the 4's column. It becomes 10 in 4's column. Keep one 1 in 4's column and bring the second 1 to the 2's column making it 10 in the 2's column.

Therefore,

In the 2's column = 10–1 = 1
In the 4's column = 1–1 = 0
In the 8's column = 0–0 = 0
Here, the result is $(0010.011)_2$

**3) Binary multiplication:**

Following are the multiplication rules :

$$0 \times 0 = 0 \; ; \; 0 \times 1 = 0 \; ; \; 1 \times 0 = 0 \; ; \; 1 \times 1 = 1$$

- For multiplication, a multiplier Q and a multiplicand M are required. Let's assume that the multiplier Q = $q_{n-1}\, q_{n-2} \ldots q_1 q_0$ and the multiplicand M = $m_{n-1}\, m_{n-2} \ldots m_1 m_0$.
- To multiply two unsigned numbers, there exists a very simple method known as the paper and pencil method.

**Example:** Consider the multiplication of two unsigned numbers 14 and 10.

| | |
|---|---|
| 1 1 1 0 (14) | multiplicand (M) |
| 1 0 1 0 (10) | multiplier (Q) |
| 0 0 0 0 | (Partial product) |
| 1 1 1 0 | (Partial product) |
| 0 0 0 0 | (Partial product) |
| 1 1 1 0 | (Partial product) |
| 1 0 0 0 1 1 0 0 | |

10001100 (140) Final product (P)

- In this method, n partial products are calculated.
- If the multiplier bit is 0, then there is no need to compute that partial product.

**Partial sum method:**

Let's take two decimal numbers $(5)_{10}$ and $(6)_{10}$:
Binary equivalent of $(5)_{10}$ = 0101
Binary equivalent of $(6)_{10}$ = 0110
Using partial sum method:
Multiplicand (M) = 0101
Multiplier (Q) = 0110

```
    0 1 0 1
  × 0 1 1 0
  ---------
    0 0 0 0
  0 1 0 1
0 1 0 1
0 0 0 0
---------
0 0 1 1 1 1 0
```

So, if we multiply two n bit numbers, then in the worst case, the space required to store the result in 2n.

**Multiplication using registers:**

Multiplying (0101) by (0110) ⟶ 0101 ×0110

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

Multiplying 0 with (0101) and performing 1 arithmetic right shift operation after addition.

Similarly, in step 2, multiplying 1 with (0101) and performing 1 arithmetic right shift after addition, in step 3, multiplying 1 with (0101) and performing 1 arithmetic right shift after addition and in step 4 multiplying 0 with (0101) and performing 1 arithmetic right shift after addition.

Step 1:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| + | 0 | 0 | 0 | 0 | | | | |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Shift:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Step 2:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| + | 0 | 1 | 0 | 1 | | | | |
| | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |

Shift:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |

Step 3:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| + | 0 | 1 | 0 | 1 | | | | |
| | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |

Shift:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |

Step 4:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| + | 0 | 0 | 0 | 0 | | | | |
| | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |

Shift:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |

(final result)

**Note:**

For every bit in the multiplier (Q). The Shift will be performed, and for every 1 in the multiplier addition will be performed.

So, the number of addition required = The number of 1's in the multiplier.

the number of shifts required = Number of bits in the multiplier.

eg: Q = 011111110

Number of addition required = 7

Number of shifts required = 9

**Fixed point sign multiplication:**

- Multiplication cannot be extended for both signed and unsigned operands because the sign bit will get disturbed during the computation.
- Hence, a separate algorithm is required for fixed-point signed multiplication, which is known as Booth's algorithm.
- Booths algorithm gives a fast response for the multiplication of fixed point signed numbers.

eg: 1

x = 10, y = 3

x * y = 30

x = 1010  y = 0011

x = 1 0 1 0

y = 0 0 1 1

```
    1 0 1 0
  1 0 1 0
0 0 0 0
0 0 0 0
---------------
0 0 1 1 1 1 0
```

0 0 1 1 1 1 0 in decimal is 30

eg: 2

x = –6, y = 3

x * y = –18

x = $(1010)_2$ (in 2's complement)

y = 3 = $(0011)_2$

x = 1 0 1 0

y = 0 0 1 1

```
    1 0 1 0
  1 0 1 0
0 0 0 0
0 0 0 0
---------------
0 0 1 1 1 1 0
```

0 0 1 1 1 1 0 in decimal is 30.

But result of x * y should be –18.

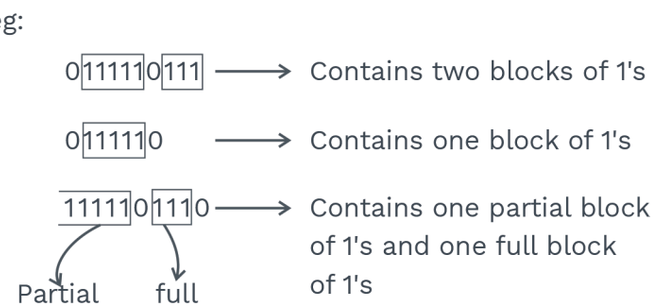
**Why need Booth's algorithm?**

It is a multiplication algorithm that is used to multiply two signed binary numbers(using 2's complement representation) in an efficient way using very less number of addition or subtraction operations.

**Booths algorithm:**

**1)** In this algorithm, a binary pattern of a multiplier is treated as blocks of 1's.

eg:



**2)** Only beginning and ending 1 of the block will result in an arithmetic operation.
**3)** Intermediate 1's in the block does not require any arithmetic operation. Hence Booths algorithm saves time as well as space.
**4)** Every full block needs 1 addition and 1 subtraction, but a partial block will need only subtraction.
**5)** Booths algorithm uses the partial sum concept, and the partial sum is arithmetically right shifted. Thus, it will protect the sign bit.

**The idea behind booth's algorithm:**
Booth's algorithm is based on one observation. Suppose we have binary number 011110 where a sequence of 1's is present.

Then this sequence (block) of 1's can be written as the difference between two numbers, where these two numbers are in the power of 2.

**Booths notation:**

- For the multiplier pattern in a booths notation, if i and j represent the beginning and ending position of the block of 1's.

The value is given as:

$$V = \sum_{m=1}^{K} \left( +2^{j_m+1} - 2^{i_m} \right)$$

Thus, if M(multiplicand) × Q(multiplier) is given where Q = 011110
then partial sum method requires 6 shifts and 4 addition operations, whereas using Booth's algorithm, we require only 1 addition and 6 shift, as:

$$M \times \overset{Q}{\boxed{0\,1\,1\,1\,1\,0}} \text{ can be written as}$$

$$= (2^5)\, M + (2^1)(-M)$$

Example:

**1)**

$$\begin{matrix} 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ 0 & 1 & 1 & 1 & 1 & 0 \end{matrix} \text{ can write as } (+ 2^5 - 2^1) = 30$$

**2)**

$$\begin{matrix} 2^3 & 2^2 & 2^1 & 2^0 \\ 0 & 1 & 1 & 0 \end{matrix} = (+ 2^3 - 2^1) = 6$$

**3)**

$$0\ 1\ 1\ 1\ 1 \ldots\ldots 1\ 1 \ldots\ldots 1\ 0 = (+ 2^{j+1} - 2^i)$$

$2^j$ $\qquad$ $2^i$

**Booths recording pattern:**

**1)** It contains (–1), 0, and (+1) as coefficients, and all weights are positive.

**2)** For all 'i' positions, the coefficient is (–1).
For all 'j+1' positions, the coefficient is (+1).
The rest of the places are 0's.

**3)** It is useful for finding:

**i)** The value of the number.

**ii)** The number of arithmetic operations as well as the number of shift operations.

e.g. 1 :

0 1 1 1 1 0 1 1 1

$j_2$ $i_2$ $j_1$ $i_1$

| 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| +1 | 0 | 0 | 0 | –1 | +1 | 0 | 0 | –1 |

$\Rightarrow$ $2^8 (+1) + 2^4 (-1) + 2^3 (+1) + 2^0 (-1)$

$\Rightarrow$ $256 - 16 + 8 - 1$

$\Rightarrow$ $247$

Number of additions required = 2

Number of subtractions required = 2

Number of shift operations required = 8 + 4 + 3 + 0 = 15

e.g. 2:

0 1 1 1 1 1 0 0

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| +1 | 0 | 0 | 0 | 0 | –1 | 0 | 0 |

$\Rightarrow \quad 2^7 (+1) + 2^2 (-1)$

$\Rightarrow \quad 128 - 4$

$\Rightarrow \quad 124$

Number of additions required = 1

Number of subtractions required = 1

Number of shift operations required = 7 + 2 = 9

Example 3 : Give Booths recording pattern for (–57).

**Sol:** (–57) in 2's complement binary number = $(1000111)_2$

| 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|
| –1 | 0 | 0 | +1 | 0 | 0 | –1 |

$\Rightarrow \quad 2^6(-1) + 2^3(+1) + 2^0(-1)$

Number of addition required = 2

Number of subtraction required = 2

Number of shift operation required = 6 + 3 + 0 = 9

Example: 4 : For which of the following, the Booth's algorithm gives a better performance?

**1)** 00011111

**2)** 10101010

**3)** 11110111

**4)** 01110111

**Sol:** Option 1), because it requires an overall less number of additions and subtractions.

Option 1):

0 0 0 11111

1 full block of 1's

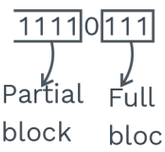
It requires 1 addition and 1 subtraction

Option 2):

10101010

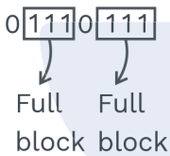
It contains 3 full block and 1 partial. Thus it requires 3 addition and 4 subtraction.

Option 3):




It contains 1 partial and 1 full block.
Thus it requires 1 addition and 2 subtraction.

Option 4):




It contains 2 full blocks.
Thus it requires 2 addition and 2 subtractions.

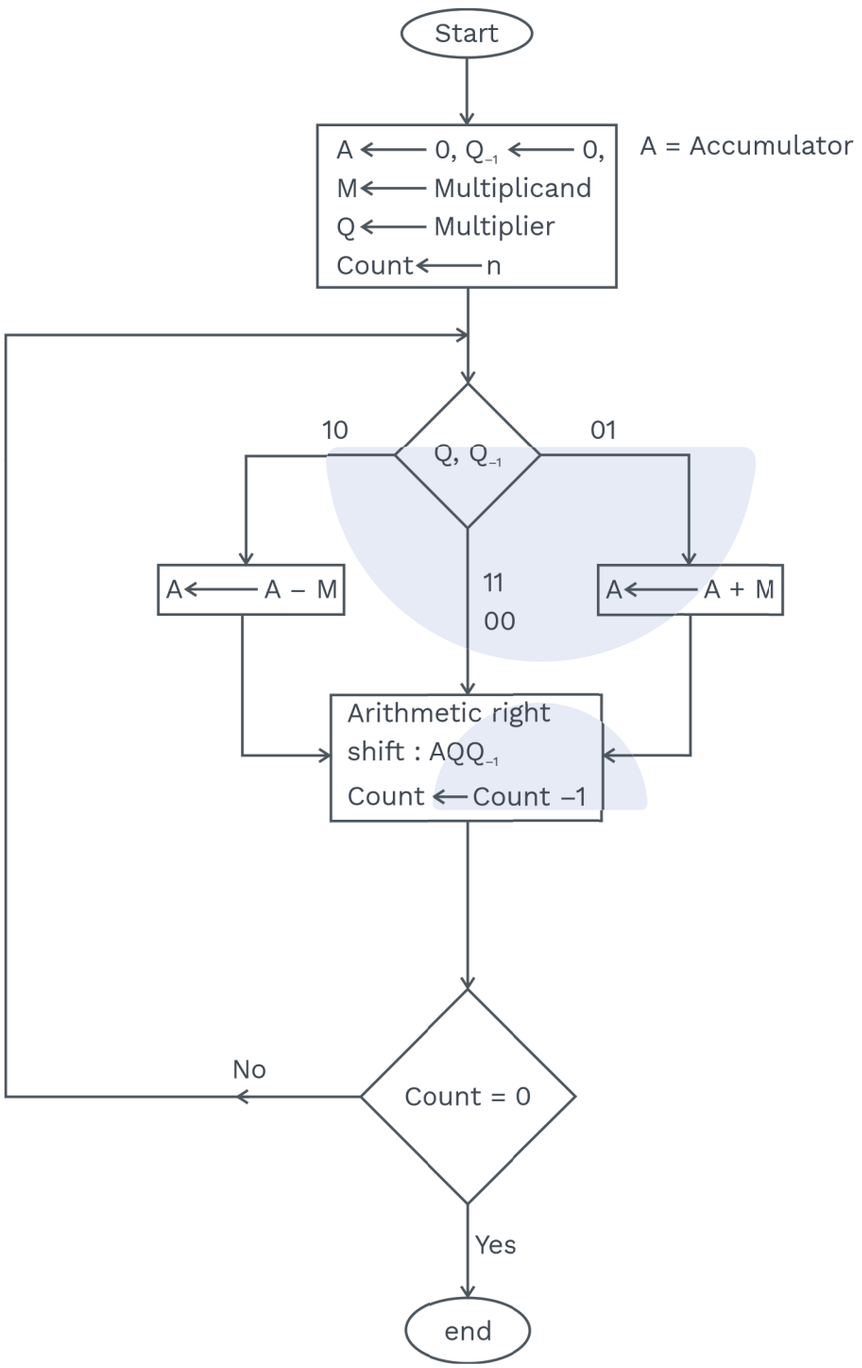



**Fig. 4.1 Flow Chart of Booth's Algorithm**

**Booth's algorithm example:**

**Example 1:** Consider the multiplication of two numbers –5 and –4 using Booth's algorithm
Here, M(multiplicand) = –5 and Q(multiplier) = –4
M in 2's complement = 1011
Q in 2's complement = 1100

- Size of accumulator register = Size of multiplicand register
- Initially, the accumulator and $Q_{-1}$ value is initialised to 0(zero).

| Operation | A | Q | $Q_{-1}$ | |
|---|---|---|---|---|
| | 0000 | 1100 | 0 | (Least significant bit of Q = 0, Thus the value of $QQ_{-1}$ = 00 performs arithmetic right shift (ARS)) |
| **1)** ARS | 0000 | 0110 | 0 | (Least significant bit of Q = 0, Thus the value of $QQ_{-1}$ = 00 perform arithmetic right shift (ARS)) |
| **2)** ARS | 0000 | 0011 | 0 | (Now, LSB of Q = 1, Thus the value of $QQ_{-1}$ = 10 perform A ¬ A–M and then ARS (arithmetic right shift)) |
| A – M = A + 2's complement of M<br>= 0000 + 0101 = 0101 | | | | |
| **3)** A – M<br>ARS | 0101<br>0010 | 0011<br>1001 | 0<br>1 | (Least significant bit of Q = 1, Thus the value of $QQ_{-1}$ = 11 perform arithmetic right shift (ARS)) |
| **4)** ARS | 0001 | 0100 | 1 | |

**Table 4.4**

Product = AQ = 00010100 = (+20)

**Example 2:** Consider the multiplication of two numbers –5 and 4 using Booth's algorithm
Here, M(multiplicand) = –5 and Q(multiplier) = 4 = $(0100)_2$
M in 2's complement = $(1011)_2$

| Operation | A | Q | $Q_{-1}$ | |
|---|---|---|---|---|
| | 0000 | 0100 | 0 | (Least significant bit of Q = 0, Thus the value of $QQ_{-1}$ = 00 perform arithmetic right shift (ARS)) |
| **1)** ARS | 0000 | 0010 | 0 | (Least significant bit of Q = 0, Thus the value of $QQ_{-1}$ = 00 perform an arithmetic right shift (ARS)) |
| **2)** ARS | 0000 | 0001 | 0 | (Now, LSB of Q = 1, Thus the value of $QQ_{-1}$ = 10 perform A ¬ A–M and then ARS (arithmetic right shift)) |
| A – M = A + 2's complement of M<br>= 0000 + 0101 = 0101 | | | | |
| **3)** A – M<br>ARS | 0101<br>0010 | 0001<br>1000 | 0<br>1 | (Now, LSB of Q = 0, Thus the value of $QQ_{-1}$ = 01 perform A ¬ A+M and then ARS (arithmetic right shift)) |
| A + M = 0010 + 1011 = 1101 | | | | |
| **4)** A + M<br>ARS | 1101<br>1110 | 1000<br>1100 | 1<br>0 | |

**Table 4.5**

Final result of multiplication = AQ = 11101100 = (–20)

**Note:**

The number of steps that need to be performed in Booth's algorithm is equal to the number of bits in the multiplier.

**Advantages and disadvantages of Booth's algorithm:**

**Advantages:**

**1)** In general, the number of operations required will be reduced.
E.g., 0111110 required 1 addition, 1 subtraction and 7 shift operation.

**2)** It is used to perform signed multiplications.
**3)** It gives better performance if negative numbers are involved.

**Disadvantages:**
When there are pairs of alternatives 0's and 1's in the multiplier, Booth's algorithm gives the worst performance.
e.g., 01010101010101

**Previous Years' Question**

The two numbers given below are multiplied using the Booth's algorithm.

| | | | | |
|---|---|---|---|---|
| Multiplicand: | 0101 | 1010 | 1110 | 1110 |
| Multiplier: | 0111 | 0111 | 1011 | 1101 |

How many additions/subtractions are required for the multiplication of the above two numbers? **(GATE IT-2008)**

**1)** 6
**2)** 8
**3)** 10
**4)** 12
**Sol: 2)**

**4) Octal addition:**
Octal addition can be performed by first converting an octal number to its binary equivalent and then applying binary arithmetic rules. Octal subtraction can also be performed by first converting it into binary and then using 1's complement or 2's complement method of binary arithmetic for subtraction.

## SOLVED EXAMPLES

**Q32** **Perform the given addition (write base as 8 for each bracket).**
**a) 25 + 13** **b) (27.5) + (74.4)**

**Sol:** **a)**

| | | |
|---|---|---|
| 25 | 010 | 101 |
| + 13 | 001 | 011 |
| 40 | 100 | 000 |

$5 + 3 = (8)_{10} = (10)_8$

$2 + 1 + 1 = (4)_{10} = (4)_8$

**b)**

$$\begin{array}{rcl} (27.5)_8 & = & 010\ \ 111\ .\ 101 \\ +(74.4)_8 & = & 111\ \ 100\ .\ 100 \\ \hline (124.1)_8 & = & 1010\ 100\ .\ 001 \end{array}$$

**Q33** **Subtract:**

**a)** $(45)_8$ from $(66)_8$
**b)** $(73)_8$ from $(25)_8$

**Sol:** Using 8-bit representation and 2's complement method.

**a)**

$$\begin{array}{rcl} (66)_8 & = & 00\ \ 110\ \ 110 \\ -(45)_8 & = & +\ 11\ \ 011\ \ 011\ (2\text{'s complement of } 45)_8 \\ \hline (21)_8 & = & (1)00\ 010\ 001 \end{array}$$

Discard

The final carry is ignored since the answer is positive.

**b)**

$$\begin{array}{rcl} (25)_8 & = & (00\ \ 010\ \ 101)_2 \\ -(73)_8 & = & +\ (11\ \ 000\ \ 101)_2\ (2\text{'s complement of } 73)_8 \\ \hline (-48)_8 & = & 11\ \ 011\ \ \ 010_2\ \ \ (\text{Answer is negative}) \end{array}$$

2's complement of $(11011010)_2 = 00100110 = -48_8$

**5) Hexadecimal arithmetic:**

Hexadecimal arithmetic rules are similar to the rules of binary or decimal number system arithmetic rules.

A hexadecimal number is first converted to its binary equivalent and then binary arithmetic rules can be used for any operation.

Hexadecimal subtraction can be performed either by converting it into its binary equivalent and then using 1's complement or 2's complement method of binary arithmetic for subtraction or by using 15's complement and 16's complement method.

## SOLVED EXAMPLES

**Q34** **Add $(6E)_{16}$ and $(C5)_{16}$**

**Sol:**

$$\begin{array}{rl} (6E)_{16} & = \ 0110 \ \ 1110 \\ +\,(C5)_{16} & = \ 1100 \ \ 0101 \\ \hline (133)_{16} & = 10011 \ \ 0011 \end{array}$$

**Q35** **Subtract $(7B)_{16}$ from $(C4)_{16}$**

**Sol:**

$$\begin{array}{rl} (C4)_{16} & = \ 1100 \ \ 0100 \\ -\,(7B)_{16} & = \ 1000 \ \ 0101 \ \text{(2's complement form of } 7B_{16}) \\ \hline (49)_{16} & = \ 1 \ 0100 \ \ 1001 \end{array}$$

Discard

The final carry is ignored since the answer is positive.

**Q36** **Subtract $(5D)_{16}$ from $(3A)_{16}$**

**Sol:**

$$\begin{array}{rl} (3A)_{16} & = \ \ 0011 \ \ 1010 \\ -\,(5D)_{16} & = +1010 \ \ 0011 \ \text{(2's complement form of } 5D)_{16} \\ \hline (-23)_{16} & = (1101 \ \ 1101)_2 \end{array}$$

$\therefore$ No carry, answer is negative.

**Q37** **Convert the following numbers with the given radix to decimal and then to binary.**

**a) $(4433)_5$** **b) $(1199)_{12}$** **c) $(5654)_7$**

**Sol:**

**a)** $(4433)_5$

**i)** $(\ )_5$ to $(\ )_{10}$

$= 5^3 \times 4 + 5^2 \times 4 + 5^1 \times 3 + 5^0 \times 3$

$= 125 \times 4 + 25 \times 4 + 5 \times 3 + 1 \times 3$

$= 500 + 100 + 15 + 3$

$= (4433)_5 \Rightarrow (618)_{10}$

**i)** $(\ )_{10}$ to $()_2$

$(618)_{10} = (1001101010)_2$

**b)** $(1199)_{12}$

**i)** $(\ )_{12}$ to $(\ )_{10}$

$= 12^3 \times 1 + 12^2 \times 1 + 12^1 \times 9 + 12^0 \times 9$

$= 1728 + 144 + 108 + 9$

$= (1199)_{12} = (1989)_{10}$

**ii)** $(\ )_{10} - (\ )_2$

$(1989)_{10} = (11111000101)_2$

**c)** $(5654)_7$

**i)** $(\ )_7$ to $(\ )_{10}$

$= 7^3 \times 5 + 7^2 \times 6 + 7^1 \times 5 + 7^0 \times 4$

$= 343 \times 5 + 49 \times 6 + 7 \times 5 + 1 \times 4$

$(5654)_7 = (2048)_{10}$

**ii)** $(\ )_{10}$ to $(\ )_2$

$(2048)_{10} = (100000000000)_2$

### Rack Your Brain

The following arithmetic operations is valid in at least one number system. Find possible base for each of the following operation.

**a)** $1234 + 5432 = 6666$

**b)** $\frac{41}{3} = 13$

**c)** $\frac{33}{3} = 11$

**d)** $44 + 32 + 14 + 23 = 223$

**e)** $\frac{302}{20} = 12.1$

**f)** $\sqrt{41} = 5$

**Rack Your Brain**

The number of bits required to assign binary roll number to a class of 60 students is -

**1)** 5 **2)** 6 **3)** 7 **4)** 8

**Previous Years' Question**

If $N^2 = (7601)_8$ where 'N' is a positive integer, then the value of N is: **(ISRO CS-2008)**

**1)** $(241)_5$

**2)** $(143)_6$

**3)** $(165)_7$

**4)** $(39)_{16}$

**Sol: 2)**

**6) BCD addition:**

BCD addition is performed in a group of 4-bits starting from LSB, and adding them column-wise. If there is no out-carry from a group and sum term is also not an illegal code, then there is no need to do anything further.

If there is an out carry from a group or the sum term is an illegal code, then $(6)_{10}$ is added to the sum term of that group and the resulting carry is added to the next group.

## SOLVED EXAMPLES

**Q38** **Perform the following decimal addition in 8421 code**

**a) 25 + 13** **b) 679.6+536.8**

**Sol:** **a)**

| | | | | |
|---|---|---|---|---|
| 25 | = | 0010 | 0101 | (25 in BCD) |
| + 13 | + | 0001 | 0011 | (13 in BCD) |
| 38 | | 0011 | 1000 | |

→ No carry, no illegal code.

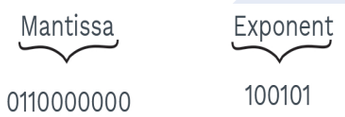
**b)**

679.6 ⇒ 0110 0111 1001 . 0110
+ 536.8 ⇒ 0101 0011 0110 . 1000
1216.4
1011 1010 1111 . 1110 ← All are illegal codes
0110 0110 0110 . 0110
10001 0000 0101 . 0100
+1 +1 + 1 + 1
0001 0010 0001 0110 . 0100
1 2 1 6 4

**Sol:** 1216.4

## 4.10 FLOATING-POINT NUMBERS

In the decimal system, very large and very small numbers are expressed by stating a number (mantissa) and an exponent of 10. Examples are $2.78 \times 10^{24}$ and $3.98 \times 10^{-26}$. Binary numbers can also be expressed in the same notation by stating a number and an exponent of 2. However, the format for a computer may be as shown below:




In this machine, the 16-bit word consists of two ports, a 10-bit mantissa, a 6-bit exponent. The mantissa is in 2's complement form. So the MSB can be thought of as the sign bit.


Floating–point is always interpreted to represent in the following form:

$$m \times r^e$$

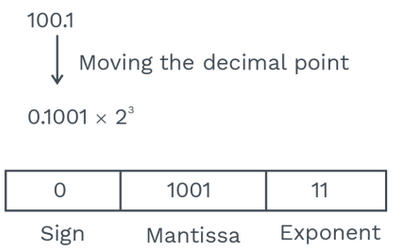
m = mantissa
e = exponent

- Mantissa can be integer or fractional.
- Exponent indicates the actual position of the decimal point.

**Why do we need floating-point?**

To represent a very large or a very small fraction of data, large memory space is required so to represent them within a less amount of memory space, floating point representation is used.

Let's represent $(4.5)_{10}$ in the above format:




$$\overset{-1\ -2\ -3\ -4}{0.1001} \times 2^3$$

$$= (2^{-1} + 2^{-4})\ 2^3$$

$$= 2^2 + 2^{-1}$$

$$= 4 + 0.5$$

$$= 4.5$$

**Normalisation:**

A floating-point number representation is not unique in the sense that, for the same number N, there are multiple representations.

**Example:** $X = 0.004 \times 10^{15}$

$X = .04 \times 10^{14}$

$X = .4 \times 10^{13}$

For hardware implementation, it is desirable to have a unique representation for a number. One way to achieve this is to represent all floating-point numbers having non-zero leading digits in their most significant position. A number expressed in this form is called normalised.

Normalisation is a process of eliminating leading zeroes from the mantissa. Its main objective is to maximise precision, given a limited number of bits.

**Example:** $X = 0.003 \times 10^7$ (not normalised number)

$X = 0.3 \times 10^5$ (normalised number)

**Note:**

- In the hexadecimal number system, mantissa should start with a non-zero leading digit (other than zero).
- In the binary number system, mantissa should start with one.

**1) Explicit normalisation:**

Move the radix point before the most significant '1'

Let's consider the number – $(100.1)_2$

↓ Moving the radix point

$0.1001 \times 2^3$ (Normalized form)


So, given a binary representation of the form:


We can convert it into decimal:

$$(-1)^S \times 0.M \times 2^E$$

**Biasing:**

**Definition**

Biasing of exponent is required in order to convert negative exponent to postive exponent

We already know, for 'n' bits number, in 2's complement form, the range of number

$$\rightarrow \left(-2^{n-1} \text{ to } 2^{n-1} - 1\right)$$

Here, $2^{(n-1)}$ is the bias for n bit exponent.

E(Unsigned biased exponent) = e + bias

Let's consider $0.101 \times 2^{-2}$ (Normalised from)

n = 5 (5 bits number)

Range of number → (–16 to + 15)

$$E = -2$$

Here, bias = $2^{n-1} = 2^{5-1} = 16$

E(Unsigned biased exponent) = –2 + 16 = 14

| S | E(5) | M |
|---|---|---|
| 0 | 01110 | 1010 |

To represent it in decimal:

$$(-1)^S \times 0.M \times 2^{E-Bias}$$

E → Biased exponent.

**Grey Matter Alert!**

Implicit Normalization gives more precission than Explicit Normalization.

**2) Implicit normalisation:**

Move the radix point after the most significant 1.

Let's consider 100.111

Let mantissa = 5 bits, exponent E = 4 bits

After normalisation the number will be $1.00111 \times 2^2$

Bias = 8, true exponent (e) = 2

$$E = 2 + 8 = 10$$

To store floating-point numbers, the register is partitioned as follows:

| S | E(4) | M(5) |
|---|---|---|
| 0 | 1010 | 00111 |

To get the corresponding decimal number, use the below expression.

$$(-1)^S \times 1.M \times 2^{E-bias}$$

## SOLVED EXAMPLES

**Q39** **Consider an IBM system which uses 32 bit register for representing the floating point number. The mantissa is implicitly normalized. The exponent is in excess –128 form and the base of the system is 2. What will be the hexadecimal pattern for storing the value $(127.5)_{10}$.**

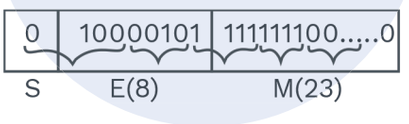
**Sol:** Value = $(-1)^S * (1.M) * 2^{E-biased}$

given the number is positive, S = 0

$$(127.5)_{10} = (1111111.1)_2 = (1.1111111)_2 \times 2^6$$

M = 1111111000.....0

E = 128 + 6 = 134 = $(10000110)_2$




**Ans.** $0 \times 37FF0000$

**Rack Your Brain**

What is the difference between 1st smallest positive and 2nd smallest positive number given


**The drawback of floating-point representation:**

1) Floating-point number distribution is not uniform since there is an unequal gap between any two consecutive numbers.
2) We can not represent a very small number (0) and a very large number (∞).
3) Floating point numbers are exposed to errors like overflow and underflow.

**Introduction to IEEE standards:**

**IEEE standards:**

1985: single, double precision

2008: half, quadruple precision

**1985 standard:**

- The base of the system is 2
- Two kinds of precision [single precision – 32 bits, double precision-64 bits]
- The floating-point number can be represented in:
  * Fractional form
  * Implicit normalised form
- Certain mantissa and exponent combinations do not represent any number (NAN → Not A Number)
- It is used to represent '0' p and '∞'

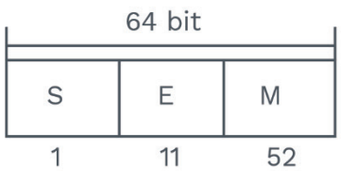
**Single precision:**


(Excess 127)

| S(1) | E(8) | M(23) | Value |
|---|---|---|---|
| 0 or 1 | 0000 0000<br>E = 0 | 000 ... 0<br>M = 0 | ± 0 |
| 0 or 1 | 1111 1111<br>E = 255 | 0000 ... 0<br>M = 0 | ± ∞ |
| 0 or 1 | 1 ≤ E ≤ 254 | M = × × ... × | Implicit normalized form<br>$(-1)^s \times 1.M \times 2^{E-127}$ |
| 1 or 0 | E = 0 | M ≠ 0 | Fractional form<br>$(-1)^s \times 0.M \times 2^{-126}$ |
| 1 or 0 | E = 255 | M ≠ 0 | NAN |

**Table 4.6**

**Double precision:**




(Excess 1023)

| S(1) | E(11) | M(52) | Value |
|---|---|---|---|
| 0 or 1 | 00000000000<br>E = 0 | 0000 ... 0<br>M = 0 | ± 0 |
| 0 or 1 | 11111111111<br>E = 2047 | 00000... 0<br>M = 0 | ± ∞ |
| 0 or 1 | 1 ≤ E ≤ 2046 | M = ××× ... × | Implicit Normalized form<br>$(-1)^s \times (1.M) \times 2^{E-1023}$ |
| 0 or 1 | E = 0 | M ≠ 0 | Fractional form<br>$(-1)^s \times 0.M \times 2^{-1022}$ |
| 0 or 1 | E= 2047 | M ≠ 0 | NAN |

**Table 4.7:**

## SOLVED EXAMPLES

**Q40** **IEEE 754 single precision format is used to store floating point numbers, what is the value of A-B?**
**Where 'A' is the smallest number represented using implicit normalized form**
**'B' is the largest number represented using fractional form.**
**A,B are all positive numbers.**

**Sol:**

(Fractional form) (Implicit normalized form)

$E = 0$, $M \neq 0$ $\quad$ $1 \leq E \leq 254$, $M = xxx...x$

Since A and B are positive numbers so, S = 0

| S | E | M |
|---|---|---|
| 1 | 8 | 23 |

32 bits

$B = (-1)^S \times (0.M) \times 2^{-126}$

$B = (-1)^0 \times (0.M) \times 2^{-126}$

$B = (0.M) \times 2^{-126}$

$0.11111....111$

GP series

$$\frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + ..... \frac{1}{2^{23}}$$

$$= \frac{\frac{1}{2}\left(1 - \frac{1}{2^{23}}\right)}{\left(1 - \frac{1}{2}\right)} = 1 - 2^{-23}$$

$B = (1 - 2^{-23}) \times 2^{-126}$

$A = (-1)^S \times 1.M \times 2^{E-127}$

$000.....0$ $\quad E = 00000001$

$= 1.0 \times 2^{1-127}$

$= 1.0 \times 2^{-126}$

$A - B = 2^{-126} - (1 - 2^{-23})\, 2^{-126}$

$= 2^{-126} [1 - 1 + 2^{-23}]$

$= 2^{-149}$

**Previous Years' Question**

Given the following binary number in 32 bit (single precision) IEEE - 754 format:
00111110011011010000000000000000
The decimal value closest to this floating-point number is: **(GATE 2017–Set-2)**

**a)** $1.45 \times 10^{1}$
**b)** $1.45 \times 10^{-1}$
**c)** $2.27 \times 10^{-1}$
**d)** $2.27 \times 10^{1}$

**Sol: c)**

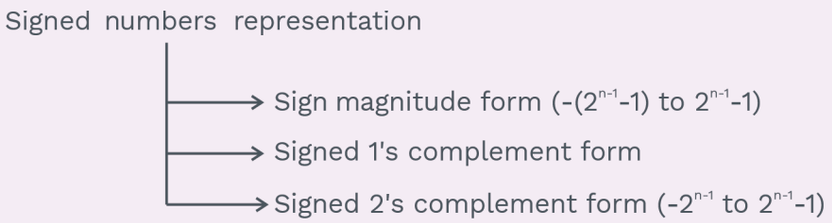
**Chapter Summary**

- Conversion from binary to decimal using positional weights.
- Conversion from decimal to binary by successive division method.
- Base-8 (octal) number system and Its conversions.
- Base-16 (hexadecimal) number system and Its conversions.
- Binary coded decimal or 8421 code.

Valid number ranges from 0-9.

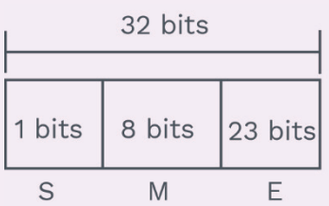
- Self-complementary code 631-1 is a self-complementary code.
- Excess-3 code - non-weighted BCD code
- Excess-3-addition → if carry occurs (add '3')
  Excess-3-addition → if carry does not occur (subtract '3')
- Gray code - Non-weighted, cyclic-unit distance code.
- Complementary number system → Diminished Radix complement $[(b^n-1)-x]$
  Complementary number system → Radix complement $[b^n-x]$

- Signed numbers representation




- Overflow: $C_{out} \neq C_{in}$ (overflow)
  $C_{out} = C_{in}$ (no overflow)
- Floating point number:


- Normalization
  - Explicit normalization
  - Implicit normalization
- IEEE Standards
  - Single precision (Excess 127)
  - Double precision (Excess 1023)
- Single-precision format:




- Double-precision format: