A number system is in general defined by the set of values and a set of rules that gives the mapping between the sequence of digits and their numerical values. A conventional number system is a non-redundant, weighted and positional. A conventional number system is fixed-radix system. When numbers are represented in machines, they have a maximum and minimum value which can be represented. If the number does not lie in this range, an overflow might occur and will produce an incorrect result.

There are two types of number representations: Fixed Point, Floating Point

The radix point "." is not stored in the register but is understood to be in a fixed position between the k most significant digits and the m least significant digits. For this reason we call such representations fixed-point representations.

To avoid the need to tell explicitly where the radix point is, we introduce the notion of **a unit in the last position (ulp)**, which is the weight of the least significant digit,

Representation of Negative Numbers: **Sign-magnitude method; Biased method; Radix complement representation method; Diminished-radix complement representation method**

**Sign Magnitude Method** - The sign and magnitude are represented separately, where the sign is represented with the first digit while the remaining n − 1 digits represent the magnitude.

| | Minimal | Maximal | Range |
|---|---|---|---|
| Positive | $0\,0 \cdots 0$ | $0\,(r-1)\cdots(r-1)$ | $[0, r^{k-1} - ulp]$ |
| Negative | $(r-1)\,(r-1)\cdots(r-1)$ | $(r-1)\,0 \cdots 0$ | $[-(r^{k-1} - ulp), 0]$ |

**Disadvantage of this method:** The operation to be performed may depend on the signs of the operands

**Biased Method** - he basic idea of this method is to let [0, Max] to represent [−Bias, Max − Bias], where the Bias is a fixed positive integer. This method sometimes is also called "excess-Bias" method.

One major disadvantage can be implied from the following:

**x + y + Bias = (x + Bias) + (y + Bias) – Bias**

**Complement Representation -** There are two alternatives for complement methods:

**Radix complement (also called 2's complement in the binary system)**

**Diminished-radix complement (called 1's complement in the binary system)**

During arithmetic operations with complement numbers,

**If X and Y have opposite signs, no overflow can occur regardless whether there is a carry-out or not.**

**If X and Y have the same sign and the sign of the result is different from that of the two operands, then an overflow occurs**

**Example 3** *Add X and Y, where* $X = -7_{10} = (11001)_2$ *and* $Y = -10_{10} = (10110)_2$.

$$
\begin{array}{rrrrrrrl}
X+Y\ : & & & & & & & \\
 & & 1 & 1 & 0 & 0 & 1 & X=-7 \\
 & + & 1 & 0 & 1 & 1 & 0 & Y=-10 \\
\hline
 & 1 & 0 & 1 & 1 & 1 & 1 & 15
\end{array}
$$

*There is a carry-out but the result is incorrect (overflow).*

**Example 4** *Add X and Y, where* $X = 7_{10} = (00111)_2$ *and* $Y = 10_{10} = (01010)_2$.

$$
\begin{array}{rrrrrrl}
X+Y\ : & & & & & & \\
 & 0 & 0 & 1 & 1 & 1 & X=7 \\
+ & 0 & 1 & 0 & 1 & 0 & Y=10 \\
\hline
 & 1 & 0 & 0 & 0 & 1 & -15
\end{array}
$$

*we do not have any carry-out but the result is incorrect (overflow).*

**Overflow can be corrected by the addition of an extra bit.**

*Convert C into nega-decimal number system (Method 2)*

| *Integer: Decimal-to-Nega-decimal* | | |
|---|---|---|
| *Dividing-by-*$(-10)$ | *Quotient* | *Remainder (keep it non-negative)* |
| $44/(-10)$ | $-4$ | $4 = x_0$ |
| $-4/(-10) = (-10+4)/(-10)$ | $1$ | $6 = x_1$ |
| $1/(-10)$ | $0$ | $1 = x_2$ |

| *Fraction: Decimal-to-Nega-decimal* | | |
|---|---|---|
| *Multiplying-by-*$(-10)$ | *Fractional part (keep it within the range)* | *Integral part* |
| $-0.3125 \times (-10)$ | $-0.875$ | $4 = x_{-1}$ |
| $-0.875 \times (-10)$ | $-0.25$ | $9 = x_{-2}$ |
| $-0.25 \times (-10)$ | $-0.5$ | $3 = x_{-3}$ |
| $-0.5 \times (-10)$ | $0$ | $5 = x_{-4}$ |

*In the second table the range is referred to an interval that a fractional number in nega-decimal form can represent. For nega-decimal system, the range is given by* $[-10/11, +1/11] = [-0.909, +0.091]$.

**For Nega-Decimal conversions, make sure that the fractional part is within the range. If not, add one to the integer part and get the fraction within range.**

**Range of NegaBinary Numbers:**

$$
\begin{cases}
F_{\max}^{-} = -(2^{-1} + 2^{-3} + 2^{-5} + 2^{-7} + \cdots) = -0.667_{10} \\
F_{\max}^{+} = (2^{-2} + 2^{-4} + 2^{-6} + 2^{-8} + \cdots) = 0.333_{10}
\end{cases}
$$

| *Integer: Decimal-to-Nega-decimal* | | |
|---|---|---|
| *Dividing-by-*$(-10)$ | *Quotient* | *Remainder (keep it non-negative)* |
| $44/(-10)$ | $-4$ | $4 = x_0$ |
| $-4/(-10) = (-10+4)/(-10)$ | $1$ | $6 = x_1$ |
| $1/(-10)$ | $0$ | $1 = x_2$ |

| *Fraction: Decimal-to-Nega-decimal* | | |
|---|---|---|
| *Multiplying-by-*$(-10)$ | *Fractional part (keep it within the range)* | *Integral part* |
| $-0.3125 \times (-10)$ | $-0.875$ | $4 = x_{-1}$ |
| $-0.875 \times (-10)$ | $-0.25$ | $9 = x_{-2}$ |
| $-0.25 \times (-10)$ | $-0.5$ | $3 = x_{-3}$ |
| $-0.5 \times (-10)$ | $0$ | $5 = x_{-4}$ |

**Multiplication of a +ve and -ve number**

$$
\begin{array}{lr|llll}
A & 1\ 0\ 1\ 1 & & & & -5 \\
X & \times\ 0\ 0\ 1\ 1 & & & & 3 \\
\hline
P^{(0)} = 0 & 0\ 0\ 0\ 0 & & & & \\
x_0 = 1 \Rightarrow \textit{Add } A & +\ 1\ 0\ 1\ 1 & & & & \\
\hline
 & 1\ 0\ 1\ 1 & & & & \\
\textit{Shift to get } P^{(1)} & 1\ 1\ 0\ 1 & 1 & & & \\
x_1 = 1 \Rightarrow \textit{Add } A & +\ 1\ 0\ 1\ 1 & & & & \\
\hline
 & 1\ 0\ 0\ 0 & 1 & & & \\
\textit{Shift to get } P^{(2)} & 1\ 1\ 0\ 0 & 0 & 1 & & \\
x_2 = 0 \Rightarrow \textit{Shift to get } P^{(3)} & 1\ 1\ 1\ 0 & 0 & 0 & 1 & \\
x_3 = 0 \Rightarrow \textit{output } P^{(3)} & 1\ 1\ 1\ 0 & 0 & 0 & 1 & = 2^{-3} \times (-15)
\end{array}
$$

**Multiplication of two -ve numbers**

**Example 3** *The multiplier is negative and the operands are in the two's complement form. Let* $X = -3$ *and* $A = -5$, *in n-bit 2's complement form, where* $n = 4$.

$$
\begin{array}{lr|llll}
A & 1\ 0\ 1\ 1 & & & & -5 \\
X & \times\ 1\ 1\ 0\ 1 & & & & -3 \\
\hline
P^{(0)} = 0 & 0\ 0\ 0\ 0 & & & & \\
x_0 = 1 \Rightarrow \textit{Add } A & 1\ 0\ 1\ 1 & & & & \\
\hline
 & 1\ 0\ 1\ 1 & & & & \\
\textit{Shift to get } P^{(1)} & 1\ 1\ 0\ 1 & 1 & & & \\
x_1 = 0 \Rightarrow \textit{Shift to get } P^{(2)} & 1\ 1\ 1\ 0 & 1 & 1 & & \\
x_2 = 1 \Rightarrow \textit{Add } A & +\ 1\ 0\ 1\ 1 & & & & \\
\hline
 & 1\ 0\ 0\ 1 & 1 & 1 & & \\
\textit{Shift to get } P^{(3)} & 1\ 1\ 0\ 0 & 1 & 1 & 1 & \\
x_3 = 1 \Rightarrow \textit{Correction (Add } -A) & +\ 0\ 1\ 0\ 1 & & & & \\
\hline
\textit{Get } P^{(3)} & 0\ 0\ 0\ 1 & 1 & 1 & 1 & = 2^{-3} \times 15
\end{array}
$$

**Fractional Division (X < D)**

**Example 4** *(restoring) Let* $X = (0.100000)_2 = 1/2$ *and* $D = (0.110)_2 = 3/4$. *Since X <*
*have*

$$
\begin{array}{lr|lll}
r_0 = X & 0\ .1\ 0\ 0 & 0\ 0\ 0 & & \\
2r_0 & 0\ 1\ .0\ 0\ 0 & 0\ 0 & & \\
\textit{Add } -D & +\ 1\ 1\ .0\ 1\ 0 & & & \\
\hline
r_1 = 2r_0 - D \geq 0 & 0\ 0\ .0\ 1\ 0 & 0\ 0 & & \textit{set } q_1 = 1 \\
2r_1 & 0\ 0\ .1\ 0\ 0 & 0 & & \\
\textit{Add } -D & +\ 1\ 1\ .0\ 1\ 0 & & & \\
\hline
2r_1 - D < 0 & 1\ 1\ .1\ 1\ 0 & 0 & & \textit{set } q_2 = 0 \\
r_2 = 2r_1 & 0\ 0\ .1\ 0\ 0 & 0 & & \\
2r_2 & 0\ 1\ .0\ 0\ 0 & & & \\
\textit{Add } -D & +\ 1\ 1\ .0\ 1\ 0 & & & \\
\hline
r_3 = 2r_2 - D \geq 0 & 0\ 0\ .0\ 1\ 0 & & & \textit{set } q_3 = 1
\end{array}
$$

*The final results are* $Q = (0.101)_2 = 5/8$ *and* $r_3 = 1/4 \Rightarrow R = r_m 2^{-m} = r_3 2^{-3} = 1/32$.

**Integer Division (32/6)**

**Condition – X < 2^(n-2) * D**

*have*

$$
\begin{array}{lr|lll}
r_0 = X & 0\ 1\ 0\ 0 & 0\ 0\ 0 & & \\
2r_0 & 0\ 1\ 0\ 0\ 0 & 0\ 0 & & \\
\textit{Add } -D & +\ 1\ 1\ 0\ 1\ 0 & & & \\
\hline
r_1 = 2r_0 - D > 0 & 0\ 0\ 0\ 1\ 0 & 0\ 0 & & \textit{set } q_1 = 1 \\
2r_1 & 0\ 0\ 1\ 0\ 0 & 0 & & \\
\textit{Add } -D & +\ 1\ 1\ 0\ 1\ 0 & & & \\
\hline
2r_1 - D < 0 & 1\ 1\ 1\ 1\ 0 & 0 & & \textit{set } q_2 = 0 \\
r_2 = 2r_1 & 0\ 0\ 1\ 0\ 0 & 0 & & \\
2r_2 & 0\ 1\ 0\ 0\ 0 & & & \\
\textit{Add } -D & +\ 1\ 1\ 0\ 1\ 0 & & & \\
\hline
r_3 = 2r_2 - D > 0 & 0\ 0\ 0\ 1\ 0 & & & \textit{set } q_3 = 1
\end{array}
$$

*The final results are* $Q = (101)_2 = 5$ *and* $R = r_3 = 2$.

**IEEE Single Precision format (32 Bits)**

An IEEE single-precision floating-point number $F$ has three parts, $s$, $e$ and $f$, as shown below.

| $s$ : sign | $e$ : exponent | $f$ : significand or mantissa |
|---|---|---|
| 1 bit | 8 bits | 23 bits |

32 bits

IEEE floating-point number $F$ can be evaluated by

$$F = (-1)^s \times 1.f \times 2^{e-127}.$$

| Value or meaning of an IEEE single-precision floating-point representation | | |
|---|---|---|
| $e = 0$ | $f = 0$ | $F = \pm 0$ |
| | $f \neq 0$ | $F$ are subnormal numbers $(= \pm 0.f \times 2^{-126})$ |
| $e = 255$ | $f = 0$ | $F = \pm\infty$ |
| | $f \neq 0$ | $F$ is NaN* (Not a Number) |
| $1 \leqslant e \leqslant 254$ | — | $F$ is an ordinary number and $F = (-1)^s \times 1.f \times 2^{e-127}$ |

* There are two types of NaN, namely signaling NaN ($f_{-1} = 0$) and quiet NaN ($f_{-1} = 1$), defined in IEEE 754-2019, where $f_{-1}$ is the leading bit in $f$ field.

**Example 1** Convert $46.5_{10}$ into IEEE single-precision standard.

Solution: First we convert the given number $46.5_{10}$ into binary of the form of (1):

$$46.5_{10} = 101110.1_2 = 1.011101 \times 2^5 = (-1)^0 \times 1.011101 \times 2^{132-127}.$$

Then it is easy to obtain

$$
\begin{cases}
s = 0 \\
e = 1000\ 0100 \\
f = 0111\ 0100\ 0000\ 0000\ 0000\ 000
\end{cases}
$$

**IEEE Double Precision format (64 Bit)**

An IEEE double-precision floating-point number $F$ has also three parts, $s$, $e$ and $f$, as shown below.

| $s$ : sign | $e$ : exponent | $f$ : significand or mantissa |
|---|---|---|
| 1 bit | 11 bits | 52 bits |

64 bits

IEEE floating-point number $F$ can be evaluated by

$$F = (-1)^s \times 1.f \times 2^{e-1023}. \qquad (2)$$

| Value or meaning of an IEEE double-precision floating-point representation | | |
|---|---|---|
| $e = 0$ | $f = 0$ | $F = \pm 0$ |
| | $f \neq 0$ | $F$ are subnormal numbers $(= \pm 0.f \times 2^{-1022})$ |
| $e = 2047$ | $f = 0$ | $F = \pm\infty$ |
| | $f \neq 0$ | $F$ is NaN (Not a Number) |
| $1 \leqslant e \leqslant 2046$ | — | $F$ is an ordinary number and $F = (-1)^s \times 1.f \times 2^{e-1023}$ |

**Example 2** Convert $46.5_{10}$ into IEEE double-precision standard.

Solution: First we convert the given number $46.5_{10}$ into binary of the form of (2):

$$46.5_{10} = 101110.1_2 = 1.011101 \times 2^5 = (-1)^0 \times 1.011101 \times 2^{1028-1023}.$$

Then it is easy to obtain

$$
\begin{cases}
s = 0 \\
e = 1000\ 0000\ 100 \\
f = 0111\ 0100\ \underbrace{00\cdots00}_{44\ zeros}
\end{cases}
$$

**Floating Point Arithmetic Operations:**

**Example 3** Convert $F_1 = 4$ and $F_2 = 3$ into IEEE floating point single precision format. Then perform floating point operations $F_1 \times F_2$ and $F_1 + F_2$.

Solution:

$$F_1 = 4 = (-1)^{S_1} \times 1.f_1 \times 2^{E_1 - 127} = (-1)^0 \times 1.0 \times 2^{129-127},$$

$$F_2 = 3 = (-1)^{S_2} \times 1.f_2 \times 2^{E_2 - 127} = (-1)^0 \times 1.1 \times 2^{128-127}.$$

Then for floating point multiplication we have

$$F_3 = F_1 \times F_2 = (-1)^0 \times (1.0 \times 1.1) \times 2^{(129-127+128)-127} = (-1)^0 \times 1.1 \times 2^{130-127}.$$

For floating point addition, since $F_1 > F_2$ we first rewrite $F_2$ such that its 2's power part is the same as that of $F_1$.

$$F_2 = (-1)^0 \times 1.1 \times 2^{128-127} = (-1)^0 \times 0.11 \times 2^{129-127}.$$

Then it follows

$$F_4 = F_1 + F_2 = (-1)^0 \times (1.0 + 0.11) \times 2^{129-127} = (-1)^0 \times 1.11 \times 2^{129-127}.$$

**For FP product (F3 = F1 x F2) - S3 = resulatant sign, f3 = (Multiplication of f1 and f2) and e3 = (e1 + e2 -127)**

**For addition/subtraction, f3 = (f1 +- f2). Make sure that the value of e1 and e2 are the same.**

Rounding Schemes:

**TRUNCATION** – Remove anything after decimal point.

| Chopping scheme with $d = 2$ | | |
|---|---|---|
| Input: $x$ | Output: $\mathrm{chop}(x)$ | Error: $\mathrm{chop}(x) - x$ |
| $\times.00$ | $\times.$ | $0$ |
| $\times.01$ | $\times.$ | $-1/4$ |
| $\times.10$ | $\times.$ | $-1/2$ |
| $\times.11$ | $\times.$ | $-3/4$ |

Implementation: Its implementation is cost free

Time delay: There is no delay incurred with this rounding scheme.

**ROUNDING TO NEAREST INTEGER** – 0.5 is attributed to the nearest integer

| Round-to-nearest scheme with $d = 2$ | | |
|---|---|---|
| Input: $x$ | Output: $\mathrm{round}(x)$ | Error: $\mathrm{round}(x) - x$ |
| $\times.00$ | $\times.$ | $0$ |
| $\times.01$ | $\times.$ | $-1/4$ |
| $\times.10$ | $\times. + 1$ | $+1/2$ |
| $\times.11$ | $\times. + 1$ | $+1/4$ |

Implementation: An implementation requires an adder and a few logic gates.

Time delay: It is equal to that of an adder of the size of the output.

**ROUND TO NEAREST EVEN INTEGER** – 0.5 is attributed to the nearest even integer

| Round-to-nearest-even scheme with $d = 2$ | | |
|---|---|---|
| Input: $x$ | Output: $\mathrm{rtne}(x)$ | Error: $\mathrm{rtne}(x) - x$ |
| $\times 0.00$ | $\times 0.$ | $0$ |
| $\times 0.01$ | $\times 0.$ | $-1/4$ |
| $\times 0.10$ | $\times 0.$ | $-1/2$ |
| $\times 0.11$ | $\times 1.$ | $+1/4$ |
| $\times 1.00$ | $\times 1.$ | $0$ |
| $\times 1.01$ | $\times 1.$ | $-1/4$ |
| $\times 1.10$ | $\times 1. + 1$ | $+1/2$ |
| $\times 1.11$ | $\times 1. + 1$ | $+1/4$ |

Implementation and time delay: slightly higher than previous method

ROM ROUNDING – For this method, L = total number of bits, d =bits after decimal point

| ROM scheme with $\ell = 3$ input bits | | |
|---|---|---|
| Input: $x$ | Output: $\mathrm{ROM}(x)$ | Error: $\mathrm{ROM}(x) - x$ |
| $\times 00.0$ | $\times 00.$ | $0$ |
| $\times 00.1$ | $\times 01.$ | $+1/2$ |
| $\times 01.0$ | $\times 01.$ | $0$ |
| $\times 01.1$ | $\times 10.$ | $+1/2$ |
| $\times 10.0$ | $\times 10.$ | $0$ |
| $\times 10.1$ | $\times 11.$ | $+1/2$ |
| $\times 11.0$ | $\times 11.$ | $0$ |
| $\times 11.1$ | $\times 11.$ | $-1/2$ |

Implementation cost: Size of the ROM used, Time delay: Decided by the ROM reading speed

**In case of ROM rounding, the final result is truncated instead of incrementing to avoid a carry chain. This needs to be done, else the system operation will be delayed. Here, we have rounded 11.1 to 11 instead of 100.**