IMPERIAL COLLEGE of ENGINEERING

(Affiliated by Rajshahi University Code: 385)

Department of CSE

Assignment no- 01

Date: 04-12-21

Day: Sunday

Assignment Name: Floating-point representations, General properties, IEEE-754, 32-bit and 64-bit formats, Denormalized numbers, NaNs and other special values, Floating-point exception handling, CRAY, Rounding methods, Floating-point operations (+, -, X, /), Catastrophic cancellation due to subtraction; introduction to the concept of condition number.

Course Code:	MATH2231
Course Name:	Numerical Methods
Semester:	Even
Part:	02

Submitted by,

Name: Teresa Jency Bala
ID: 1938520113
Part-02, Even Semester
Dept. of CSE
Imperial College of Engineering, Khulna

Submitted to,

Md. Abul Ala Walid
Lecturer, Dept. of CSE

Floating-point representations:

A floating point number is a number which contains decimal point. This causes the number to have fractional quantities.

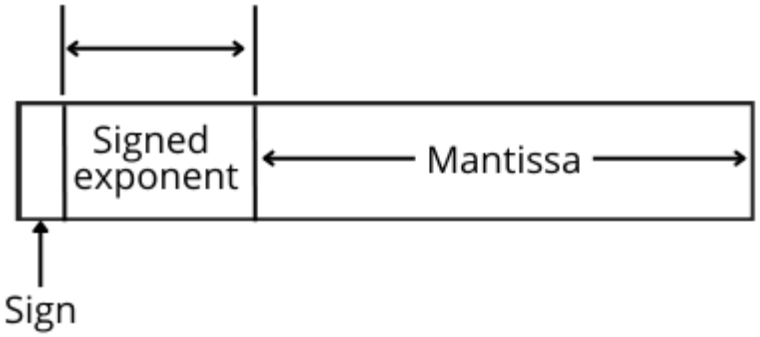
It could be in a form of 3.14159265358, $3 * 10^{-4}$ etc.

The integer part is called exponent or characteristics and the fractional part is called mantissa.

If we represent as $M*B^e$ then,

M = Mantissa, B = the base of the number system being used and e = the exponent.

For example: the number 156.78 could be represented as $0.15678 * 10^3$ in a floating-point base-10 system.



The floating point number could be stored in a word. The first bit reserved for the sign, the next series of bits for the signed exponent, and the last bits for mantissa.

The mantissa is normalized if it has leading zero digits.

For example: 1/34 = 0.029411765 if stored in in a floating point base-10 system and allowed only four decimal places to be stored, then it would be $0.02941 * 10^0$.

The 0 on the right of the decimal is not necessary, so we can normalize it by removing the leading zero by multiplying the mantissa by 10 and lowering exponent by 1 as, $0.2941*10^{-1}$.

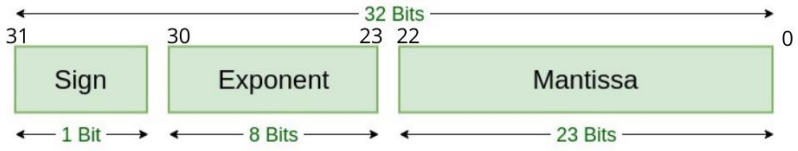
This method is known as Scientific notation.

General properties:

1. A floating-point number is a rational number, because it can be represented as one integer divided by another; for example 1.45×10^3 is (145/100)×1000 or 145,000/100.
2. Some common rules of arithmetic are not always valid when applied to floating-point numbers.
3. Floating-point numbers are all rational numbers. But they are only a subset of the rational numbers. So, not all rational numbers are floating-point numbers
4. There are only a finite number of floating-point numbers.
5. The decimal equivalent of any finite floating-point value contains a finite number of non-zero digits.

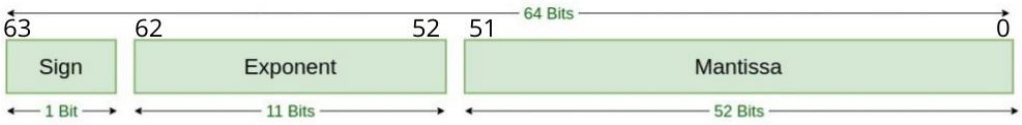
IEEE-754, 32-bit and 64-bit formats for floating point representation:

a) Single precision format: For a 32-bit register. The component



Single Precision
IEEE 754 Floating-Point Standard

b) Double precision format:



Double Precision
IEEE 754 Floating-Point Standard

Normalized number:

A normalized number is a number for which both the exponent (including bias) and the most significant bit of the mantissa are non-zero. For such numbers, all the bits of the mantissa contribute to the precision of the representation. A number is normalized when it is written in the form of $a \times 10n$ where $1 \leq a < 10$ without leading zeros in a. This is the standard form of scientific notation. An alternative style is to have the first non-zero digit after the decimal point.

Normalized decimal number: 1. Mantissa X 2^{exponent}

As examples, the number 528.123 in normalized form is 5.28 X 10^2

The number −0.000123 in normalized form is -1.23 X 10^{-4}

Clearly, any non-zero real number can be normalized.

Denormalized numbers

Denormalized numbers represent values between the Underflow limits and zero.

i.e. For single precision we have: ± 0.F X 2^{-126}

Thus denormalized single-precision numbers can be in the range (plus or minus) $2^{-126} \times 2^{-22} = 2^{-148}$ to $(1-2^{-22}) \times 2^{-126}$ inclusive.

Denormalized double-precision numbers can be in the range (plus or minus) $2^{-1022} \times 2^{-51} = 2^{-1073}$ to $(1-2^{-51}) \times 2^{-1022}$ inclusive.

Both positive and negative zero values exist, but they are treated the same during floating point calculations.

An Exponent of All 0's is used to represent Zero and Denormalized number, while All 1's is used to represent Infinities and Not-a-Number (NaNs).

Denormal Format:

Denormal floating point numbers are stored without the implicit leading one bit,

$X = \pm 0.f * 2^{-emax+1}$. The fraction f satisfies, $0 \leq f < 1$

NaNs (Not a number) and other special values:

NaNs (Not a number) are used to represent the results of operations which have no mathematical interpretation. Example: 0/0, +Infinity, -Infinity, 0 X Infinity, Square root of a (–ve) number.

Operations with a NaN operand yield either a NaN result (quiet Nan operand) or an exception (signaling NaN operand).

In the Cray format floating point, there is a concept of a positive or negative indefinite number. With IEEE, there is a different concept of "Not-a-Number" or NaN. Some operations of floating-point arithmetic are invalid, such as taking the square root of a negative number. The act of reaching an invalid result is called a floating-point exception. An exceptional result is represented by a special code called a NaN, for "Not a Number". All NaNs in IEEE 754-1985 have this format:

sign = either 0 or 1.

biased exponent = all 1 bits.

fraction = anything except all 0 bits (since all 0 bits represents infinity).

For example, a bit-wise IEEE 754 single precision (32-bit) NaN would be

s111 1111 1xxx xxxx xxxx xxxx xxxx xxxx

Floating-point exception handling:

The Institute of Electrical and Electronics Engineers (IEEE) defines a standard for floating-point exceptions called the IEEE Standard for Binary Floating-Point Arithmetic (IEEE 754). This standard defines five types of floating-point exception that must be signaled when detected:

- Invalid operation
- Division by zero
- Overflow
- Underflow
- Inexact calculation

Cray Floating point numbers:

In the Cray format floating point, there is a concept of a positive or negative indefinite number. The Cray-1 system has a sign bit, 15 bits of excess-16, 384 exponent, and 48 bits of mantissa using the more common sign-magnitude format for floating-point numbers. The Cray-1 uses 64 bits for floating point and integer number that allowed both for a very wide range of numbers to be represented at high precision.

Rounding methods

In floating point arithmetic, two extra bits are used to the far right of the significant, called the guard and round bits. At the end of the arithmetic calculation, these bits are rounded off. We always round towards the closer digit (i.e. 0.00-0.49 -> 0 and 0.51-0.99 -> 1).

IEEE 754 defines 4 rounding modes to determine how the extra two two bits are used:

1) **Towards +∞** rounds "up": $1.50_{10} \rightarrow 2_{10}$ $-1.50_{10} \rightarrow -1_{10}$ $2.50_{10} \rightarrow 3_{10}$	2) **Towards -∞** rounds "down": $1.50_{10} \rightarrow 1_{10}$ $-1.50_{10} \rightarrow -2_{10}$ $2.50_{10} \rightarrow 2_{10}$	3) **Truncate** rounds towards 0: $1.50_{10} \rightarrow 1_{10}$ $-1.50_{10} \rightarrow -1_{10}$ $2.50_{10} \rightarrow 2_{10}$	4) **Unbiased** rounds to even $1.50_{10} \rightarrow 2_{10}$ $-1.50_{10} \rightarrow -2_{10}$ $2.50_{10} \rightarrow 2_{10}$

Floating-point operations (+, -, X, /)

ADDITION

Example on decimal value given in scientific notation:

3.25×10^{3}
$+ 2.63 \times 10^{-1}$ First step: align decimal points

3.25×10^{3}
$+0.000263 \times 10^{3}$ Second step: add

3.250263×10^{3}

SUBTRACTION

Same as addition as far as alignment of radix points
Then the algorithm for subtraction of sign magnitude numbers takes over.

MULTIPLICATION
Floating Point Multiplication
N1 x N2 = (M1 x 10E1) x (M2 x 10E2)
= (M1 x M2) x (10E1 x 10E2)
= (M1 x M2) x (10E1+E2)
i.e. We multiply the Mantissas and Add the Exponents

Example on decimal values given in scientific notation:

2.0×10^{1}
$+ \ 0.5 \times 10^{2}$
_____
Algorithm: multiply mantissas and add exponents

2.0×10^{1}
$+ \ 0.5 \times 10^{2}$
_____
1.00×10^{3}
We must also normalize the result, so the final answer = 1.00×10^{3}

Division:

$X1 = (-1)^{S1} M1 \times 10^{E1}$, $X2 = (-1)^{S2} (M2 \times 2E2)$

$X3 = (X1/X2)$

$= (-1)^{S1} (M1 \times 10^{E1}) / (-1)^{S2} (M2 \times 10^{E2})$

$= (-1)^{S3} (M1/M2) \ X \ 10^{(E1-E2)}$

Catastrophic cancellation due to subtraction:

Catastrophic cancellation occurs when the operands are subject to rounding errors.

To compress infinite number of real numbers into a finite number of bits requires an approximate representation. Although there are infinitely many integers, in most programs the result of integer computations can be stored in 32 bits. In contrast, given any fixed number of bits, most calculations with real numbers will produce quantities that cannot be exactly represented using that many bits. Therefore the result of a floating-point calculation must often be rounded in order to fit back into its finite representation. This rounding error is the characteristic feature of floating-point computation.

Use the same floating-point system as in the previous example to compute b^2-4ac for b = 3.34, a = 1.22, and c = 2.28. The exact value is 3.34^2 - (4 X 1.22 X 2.28) = 0.0292 = 2.92×10^{-2}, and this exact value is representable in our floating-point system. Look at how the value is calculated, though:

$b^2 = (3.34)^2 = 11.1556 \approx 1.12 \times 10^1$
$4ac = 4 \times 1.22 \times 2.28 = 4.88 \times 2.28 = 11.1264 \approx 1.11 \times 10^1$

$b^2 - 4ac \approx 1.12 \times 10^1 - 1.11 \times 10^1 = 0.01 \times 10^1 = 1.00 \times 10^{-1} = 0.1$

Compared to our exact answer of 2.92×10^{-2}=0.0292, this has a relative error of

$$\frac{|0.0292 - 0.1|}{0.0292} = 2.424 \ldots > 240\%$$

Subtracting two floating-point numbers that are very close together leaves very few significant digits — a great deal of information is lost. Since the true value is very small, the round-off error becomes much more significant, and sometimes becomes much larger than the value being computed.

The expression $b^2 - 4ac$ crops up in the solution to the quadratic equation $ax^2 + bx + c = 0$. The general form of the solution for the two roots x_1 and x_2 is

$$x1 = \frac{-b+\sqrt{b^2-4ac}}{2a}, \; x2 = \frac{-b-\sqrt{b^2-4ac}}{2a}$$

We may not have to worry about the large relative error in $b^2 - 4ac$, since it may be small in absolute value compared to −b. Here's a case where computing x1 (using the values that lead to catastrophic cancellation above) gives a fairly acceptable value using floating-point operations:

$$x_1 = \frac{-3.34 + \sqrt{0.1}}{2 \times 1.22} = \frac{-3.34 + 0.316}{2.44} = -1.24$$

$x_1 = \frac{-3.34 + \sqrt{0.0292}}{2 \times 1.22} = \frac{-3.34 + 0.171}{2.44} = -1.30$

$\frac{|1.3-1.24|}{1.3} X100$ =4.6% for a relative error of less than 5%.

Introduction to the concept of condition number:

Condition number

The ratio of the size of change in the output of a function to the size of change in the minute input that produced it.

The condition number is the maximum error magnification:

(Forward error)≤ (conditional number) * (backward error)

If the condition number is big, then small changes in the problem (backward error) lead to big error in the solution (forward error). That is, how much small changes in the problem are magnified. A problem with large condition number is ill-conditioned. The condition number of root finding is proportional to $\frac{1}{f'(r)}$. Condition can be different points.

If r is the exact solution to f(x) =0 and r_a is our approximation solution, then

{ Data that defines The problem: f (x) } ⟶ { Equation solver } ⟶ { R such that f(r) = 0 }

Backward error $|f(r_a)|$ forward error: $|r-r_a|$

The forward error is the difference between the exact answer and the computed answer $|r - r_a|$

The computed answer r_a is the solution to some problem $\bar{f}$ (x) = 0. Thus $\bar{f}$ $(r_a) = 0$ with $\bar{f} \approx f$

The backward error is the difference between the original problem and the perturbed problem: $|f(r) - \bar{f}(r_a)| = |\bar{f}(r_a)|$. This says how far we are from the problem that the algorithm actually solved.