Chapter 01.05
Floating Point Representation

After reading this chapter, you should be able to:

1. *convert a base-10 number to a binary floating point representation,*
2. *convert a binary floating point number to its equivalent base-10 number,*
3. *understand the IEEE-754 specifications of a floating point representation in a typical computer,*
4. *calculate the machine epsilon of a representation.*

Consider an old time cash register that would ring any purchase between 0 and 999.99 units of money. Note that there are five (not six) working spaces in the cash register (the decimal number is shown just for clarification).

Q: How will the smallest number 0 be represented?

A: The number 0 will be represented as

0	0	0	.	0	0

Q: How will the largest number 999.99 be represented?

A: The number 999.99 will be represented as

9	9	9	.	9	9

Q: Now look at any typical number between 0 and 999.99, such as 256.78. How would it be represented?

A: The number 256.78 will be represented as

2	5	6	.	7	8

Q: What is the smallest change between consecutive numbers?

A: It is 0.01, like between the numbers 256.78 and 256.79.

Q: What amount would one pay for an item, if it costs 256.789?

A: The amount one would pay would be rounded off to 256.79 or chopped to 256.78. In either case, the maximum error in the payment would be less than 0.01.

Q: What magnitude of relative errors would occur in a transaction?

A: Relative error for representing small numbers is going to be high, while for large numbers the relative error is going to be small.

For example, for 256.786, rounding it off to 256.79 accounts for a round-off error of $256.786 - 256.79 = -0.004$. The relative error in this case is

$$\varepsilon_t = \frac{-0.004}{256.786} \times 100$$
$$= -0.001558\% .$$

For another number, 3.546, rounding it off to 3.55 accounts for the same round-off error of $3.546 - 3.55 = -0.004$. The relative error in this case is

$$\varepsilon_t = \frac{-0.004}{3.546} \times 100$$
$$= -0.11280\% .$$

Q: If I am interested in keeping relative errors of similar magnitude for the range of numbers, what alternatives do I have?

A: To keep the relative error of similar order for all numbers, one may use a floating-point representation of the number. For example, in floating-point representation, a number

256.78 is written as $+2.5678 \times 10^2$,

0.003678 is written as $+3.678 \times 10^{-3}$, and

-256.789 is written as -2.56789×10^2.

The general representation of a number in base-10 format is given as

$$sign \times mantissa \times 10^{\text{exponent}}$$

or for a number y,

$$y = \sigma \times m \times 10^e$$

Where

$\sigma = \text{sign of the number}, +1 \text{ or } -1$

$m = \text{mantissa}, 1 \le m < 10$

$e = \text{integer exponent (also called ficand)}$

Let us go back to the example where we have five spaces available for a number. Let us also limit ourselves to positive numbers with positive exponents for this example. If we use the same five spaces, then let us use four for the mantissa and the last one for the exponent. So the smallest number that can be represented is 1 but the largest number would be 9.999×10^9. By using the floating-point representation, what we lose in accuracy, we gain in the range of numbers that can be represented. For our example, the maximum number represented changed from 999.99 to 9.999×10^9.

What is the error in representing numbers in the scientific format? Take the previous example of 256.78. It would be represented as 2.568×10^2 and in the five spaces as

2	5	6	8	2

Another example, the number 576329.78 would be represented as 5.763×10^5 and in five spaces as

5	7	6	3	5

So, how much error is caused by such representation. In representing 256.78, the round off error created is $256.78 - 256.8 = -0.02$, and the relative error is

$$\varepsilon_t = \frac{-0.02}{256.78} \times 100 = -0.0077888\%,$$

In representing 576329.78, the round off error created is $576329.78 - 5.763 \times 10^5 = 29.78$, and the relative error is

$$\varepsilon_t = \frac{29.78}{576329.78} \times 100 = 0.0051672\%.$$

What you are seeing now is that although the errors are large for large numbers, but the relative errors are of the same order for both large and small numbers.

Q: How does this floating-point format relate to binary format?
A: A number y would be written as

$$y = \sigma \times m \times 2^e$$

Where

σ = sign of number (negative or positive – use 0 for positive and 1 for negative),
m = mantissa, $(1)_2 \le m < (10)_2$, that is, $(1)_{10} \le m < (2)_{10}$, and
e = integer exponent.

Example 1

Represent $(54.75)_{10}$ in floating point binary format. Assuming that the number is written to a hypothetical word that is 9 bits long where the first bit is used for the sign of the number, the second bit for the sign of the exponent, the next four bits for the mantissa, and the next three bits for the exponent,

Solution

$$(54.75)_{10} = (110110.11)_2 = (1.1011011)_2 \times 2^{(5)_{10}}$$

The exponent 5 is equivalent in binary format as

$$(5)_{10} = (101)_2$$

Hence

$$(54.75)_{10} = (1.1011011)_2 \times 2^{(101)_2}$$

The sign of the number is positive, so the bit for the sign of the number will have zero in it.

$$\sigma = 0$$

The sign of the exponent is positive. So the bit for the sign of the exponent will have zero in it.

The mantissa

$$m = 1011$$

(There are only 4 places for the mantissa, and the leading 1 is not stored as it is always expected to be there), and
the exponent

$$e = 101.$$

we have the representation as

0	0	1	0	1	1	1	0	1

Example 2

What number does the below given floating point format

0	1	1	0	1	1	1	1	0

represent in base-10 format. Assume a hypothetical 9-bit word, where the first bit is used for the sign of the number, second bit for the sign of the exponent, next four bits for the mantissa and next three for the exponent.

Solution

Given

Bit Representation	Part of Floating point number
0	Sign of number
1	Sign of exponent
1011	Magnitude of mantissa
110	Magnitude of exponent

The first bit is 0, so the number is positive.
The second bit is 1, so the exponent is negative.
The next four bits, 1011, are the magnitude of the mantissa, so

$$m=(1.1011)_2=\left(1\times 2^0+1\times 2^{-1}+0\times 2^{-2}+1\times 2^{-3}+1\times 2^{-4}\right)_{10}=(1.6875)_{10}$$

The last three bits, 110, are the magnitude of the exponent, so

$$e=(110)_2=\left(1\times 2^2+1\times 2^1+0\times 2^0\right)_{10}=(6)_{10}$$

The number in binary format then is

$$(1.1011)_2\times 2^{-(110)_2}$$

The number in base-10 format is

$$=1.6875\times 2^{-6}$$
$$=0.026367$$

Example 3

A machine stores floating-point numbers in a hypothetical 10-bit binary word. It employs the first bit for the sign of the number, the second one for the sign of the exponent, the next four for the exponent, and the last four for the magnitude of the mantissa.

a) Find how 0.02832 will be represented in the floating-point 10-bit word.
b) What is the decimal equivalent of the 10-bit word representation of part (a)?

Solution

a) For the number, we have the integer part as 0 and the fractional part as 0.02832
Let us first find the binary equivalent of the integer part

Integer part $(0)_{10}=(0)_2$

Now we find the binary equivalent of the fractional part

Fractional part: $\underline{.02832\times 2}$

$\underline{0.05664\times 2}$

$\underline{0.11328\times 2}$

$\underline{0.22656\times 2}$

$$\begin{aligned}
&\underline{0.45312\times 2}\\
&\underline{0.90624\times 2}\\
&\underline{1.81248\times 2}\\
&\underline{1.62496\times 2}\\
&\underline{1.24992\times 2}\\
&\underline{0.49984\times 2}\\
&\underline{0.99968\times 2}\\
&\underline{1.99936}
\end{aligned}$$

Hence

$$\begin{aligned}
&(0.02832)_{10} \cong (0.00000111001)_2\\
&=(1.11001)_2 \times 2^{-6}\\
&\cong (1.1100)_2 \times 2^{-6}
\end{aligned}$$

The binary equivalent of exponent is found as follows

	Quotient	Remainder
6/2	3	$0 = a_0$
3/2	1	$1 = a_1$
1/2	0	$1 = a_2$

So

$$(6)_{10} = (110)_2$$

So

$$\begin{aligned}
(0.02832)_{10} &= (1.1100)_2 \times 2^{-(110)_2}\\
&= (1.1100)_2 \times 2^{-(0110)_2}
\end{aligned}$$

Part of Floating point number	Bit Representation
Sign of number is positive	0
Sign of exponent is negative	1
Magnitude of the exponent	0110
Magnitude of mantissa	1100

The ten-bit representation bit by bit is

0	1	0	1	1	0	1	1	0	0

b) Converting the above floating point representation from part (a) to base 10 by following Example 2 gives

$$\begin{aligned}
&(1.1100)_2 \times 2^{-(0110)_2}\\
&=\left(1\times 2^0 + 1\times 2^{-1} + 1\times 2^{-2} + 0\times 2^{-3} + 0\times 2^{-4}\right) \times 2^{-\left(0\times 2^3 + 1\times 2^2 + 1\times 2^1 + 0\times 2^0\right)}\\
&=(1.75)_{10} \times 2^{-(6)_{10}}\\
&= 0.02734375
\end{aligned}$$

Q: How do you determine the accuracy of a floating-point representation of a number?

A: The machine epsilon, $\in_{mach}$ is a measure of the accuracy of a floating point representation and is found by calculating the difference between 1 and the next number that can be represented. For example, assume a 10-bit hypothetical computer where the first bit is used for the sign of the number, the second bit for the sign of the exponent, the next four bits for the exponent and the next four for the mantissa.

We represent 1 as

0	0	0	0	0	0	0	0	0	0

and the next higher number that can be represented is

0	0	0	0	0	0	0	0	0	1

The difference between the two numbers is

$$(1.0001)_2 \times 2^{(0000)_2} - (1.0000)_2 \times 2^{(0000)_2}$$
$$= (0.0001)_2$$
$$= (1 \times 2^{-4})_{10}$$
$$= (0.0625)_{10}.$$

The machine epsilon is

$$\in_{mach} = 0.0625.$$

The machine epsilon, $\in_{mach}$ is also simply calculated as two to the negative power of the number of bits used for mantissa. As far as determining accuracy, machine epsilon, $\in_{mach}$ is an upper bound of the magnitude of relative error that is created by the approximate representation of a number (See Example 4).

Example 4

A machine stores floating-point numbers in a hypothetical 10-bit binary word. It employs the first bit for the sign of the number, the second one for the sign of the exponent, the next four for the exponent, and the last four for the magnitude of the mantissa. Confirm that the magnitude of the relative true error that results from approximate representation of 0.02832 in the 10-bit format (as found in previous example) is less than the machine epsilon.

Solution

From Example 2, the ten-bit representation of 0.02832 bit-by-bit is

0	1	0	1	1	0	1	1	0	0

Again from Example 2, converting the above floating point representation to base-10 gives

$$(1.1100)_2 \times 2^{-(0110)_2}$$
$$= (1.75)_{10} \times 2^{-(6)_{10}}$$
$$= (0.02734375)_{10}$$

The absolute relative true error between the number 0.02832 and its approximate representation 0.02734375 is

$$|\varepsilon_t| = \left|\frac{0.02832 - 0.02734375}{0.02832}\right|$$
$$= 0.034472$$

which is less than the machine epsilon for a computer that uses 4 bits for mantissa, that is,

$$\begin{aligned}\varepsilon_{mach} &= 2^{-4} \\ &= 0.0625\end{aligned}.$$

Q: How are numbers actually represented in floating point in a real computer?

A: In an actual typical computer, a real number is stored as per the IEEE-754 (Institute of Electrical and Electronics Engineers) floating-point arithmetic format. To keep the discussion short and simple, let us point out the salient features of the single precision format.

- A single precision number uses 32 bits.
- A number y is represented as

$$y = \sigma \times (1.a_1 a_2 \cdots a_{23}) \cdot 2^e$$

where

σ = sign of the number (positive or negative)

a_i = entries of the mantissa, can be only 0 or 1, $i = 1,..,23$

e =the exponent

Note the 1 before the radix point.

- The first bit represents the sign of the number (0 for positive number and 1 for a negative number).
- The next eight bits represent the exponent. Note that there is no separate bit for the sign of the exponent. The sign of the exponent is taken care of by normalizing by adding 127 to the actual exponent. For example in the previous example, the exponent was 6. It would be stored as the binary equivalent of $127 + 6 = 133$. Why is 127 and not some other number added to the actual exponent? Because in eight bits the largest integer that can be represented is $(11111111)_2 = 255$, and halfway of 255 is 127. This allows negative and positive exponents to be represented equally. The normalized (also called biased) exponent has the range from 0 to 255, and hence the exponent e has the range of $-127 \le e \le 128$.
- If instead of using the biased exponent, let us suppose we still used eight bits for the exponent but used one bit for the sign of the exponent and seven bits for the exponent magnitude. In seven bits, the largest integer that can be represented is $(1111111)_2 = 127$ in which case the exponent e range would have been smaller, that is, $-127 \le e \le 127$. By biasing the exponent, the unnecessary representation of a negative zero and positive zero exponent (which are the same) is also avoided.
- Actually, the biased exponent range used in the IEEE-754 format is not 0 to 255, but 1 to 254. Hence, exponent e has the range of $-126 \le e \le 127$. So what are $e = -127$ and $e = 128$ used for? If $e = 128$ and all the mantissa entries are zeros, the number is $\pm\infty$ (the sign of infinity is governed by the sign bit), if $e = 128$ and the mantissa entries are not zero, the number being represented is Not a Number (NaN). Because of the leading 1 in the floating point representation, the number zero cannot be represented exactly. That is why the number zero (0) is represented by $e = -127$ and all the mantissa entries being zero.
- The next twenty-three bits are used for the mantissa.
- The largest number by magnitude that is represented by this format is

$$\left(1\times 2^0 + 1\times 2^{-1} + 1\times 2^{-2} + \cdots + 1\times 2^{-22} + 1\times 2^{-23}\right)\times 2^{127} = 3.40\times 10^{38}$$

The smallest number by magnitude that is represented, other than zero, is

$$\left(1\times 2^{0}+0\times 2^{-1}+0\times 2^{-2}+\cdots+0\times 2^{-22}+0\times 2^{-23}\right)\times 2^{-126}=1.18\times 10^{-38}$$

- Since 23 bits are used for the mantissa, the machine epsilon,

$$\begin{aligned}\in_{mach} &= 2^{-23}\\ &=1.19\times 10^{-7}\end{aligned}.$$

Q: How are numbers represented in floating point in double precision in a computer?
A: In double precision IEEE-754 format, a real number is stored in 64 bits.

- The first bit is used for the sign,
- the next 11 bits are used for the exponent, and
- the rest of the bits, that is 52, are used for mantissa.

Can you find in double precision the

- range of the biased exponent,
- smallest number that can be represented,
- largest number that can be represented, and
- machine epsilon?

INTRODUCTION TO NUMERICAL METHODS	
Topic	Floating Point Representation
Summary	Textbook notes on floating point representation
Major	General Engineering
Authors	Autar Kaw
Date	December 23, 2009
Web Site	http://numericalmethods.eng.usf.edu