Report 6
Report Submitted by: Saurabh Sharma
Class Roll No.: 36
Course: Introduction to Programming

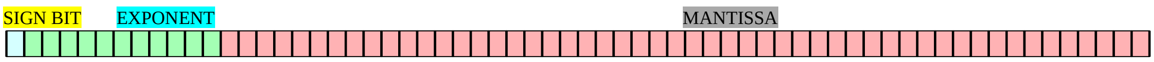
FLOATING-POINT REPRESENTATION IN COMPUTER MEMORY

1. INTRODUCTION

We expect computers to be entirely accurate and precise with numbers. To address such problems, numerous ways to represent floating-point numbers in computer memory have been introduced. The IEEE Standard for Floating-Point Arithmetic (IEEE 754), introduced in 1965 by Institute of Electrical and Electronics Engineers (IEEE) is the most efficient technical format for floating-point arithmetic in most cases.

2. FLOATING-POINT REPRESENTATION

As programmers, we are aware of variable datatypes *Single* and *Double*. As of IEEE 754 Standard, A single precision floating point number is a 32-bit binary number, consisting of Sign Bit (1 bit), Exponent (8 bits), and Mantissa (23 bits). Whereas, a double precision floating point number is a 64-bit binary number, again, consisting of Sign Bit (1 bit), Exponent (11 bits), and Mantissa (52 bits). A general representation for a 64-bit double-precision floating-point number can be depicted as the figure below.



IEEE 754 Double-Precision 64-bit Floating-Point Notation

2.1. CALCULATION

Calculating the IEEE 754 Double Precision 64-bit notation for a double could be a pretty tedious and long work, if done manually. For instance, let us represent 3.14 as an IEEE 754 Double Precision 64-bit Notation.

Step 1. First, convert to the binary (base 2) the integer part, i.e., 3. Divide the number repeatedly by 2. Keep track of each remainder. We stop when we get a quotient that is equal to zero.

division = quotient + remainder.
$3 \div 2 = 1 + 1$
$1 \div 2 = 0 + 1$

Step 2. Construct the base 2 representation of the integer part of the number.
Take all the remainders starting from the bottom of the list constructed above.

$$3_{(10)} = 11_{(2)}$$

Step 3. Convert to the binary (base 2) the fractional part: 0.14. Multiply it repeatedly by 2. Keep track of each integer part of the results. Stop when we get a fractional part that is equal to zero.

#) multiplying = integer + fractional part;

- 1) $0.14 \times 2 = 0 + 0.28$;
- 2) $0.28 \times 2 = 0 + 0.56$;
- 3) $0.56 \times 2 = 1 + 0.12$;

- 4) $0.12 \times 2 = 0 + 0.24$;
- 5) $0.24 \times 2 = 0 + 0.48$;
- 6) $0.48 \times 2 = 0 + 0.96$;
- 7) $0.96 \times 2 = 1 + 0.92$;
- 8) $0.92 \times 2 = 1 + 0.84$;
- 9) $0.84 \times 2 = 1 + 0.68$;
- 10) $0.68 \times 2 = 1 + 0.36$;
- 11) $0.36 \times 2 = 0 + 0.72$;
- 12) $0.72 \times 2 = 1 + 0.44$;
- 13) $0.44 \times 2 = 0 + 0.88$;
- 14) $0.88 \times 2 = 1 + 0.76$;
- 15) $0.76 \times 2 = 1 + 0.52$;
- 16) $0.52 \times 2 = 1 + 0.04$;
- 17) $0.04 \times 2 = 0 + 0.08$;
- 18) $0.08 \times 2 = 0 + 0.16$;
- 19) $0.16 \times 2 = 0 + 0.32$;
- 20) $0.32 \times 2 = 0 + 0.64$;
- 21) $0.64 \times 2 = 1 + 0.28$;
- 22) $0.28 \times 2 = 0 + 0.56$;
- 23) $0.56 \times 2 = 1 + 0.12$;
- 24) $0.12 \times 2 = 0 + 0.24$;
- 25) $0.24 \times 2 = 0 + 0.48$;
- 26) $0.48 \times 2 = 0 + 0.96$;
- 27) $0.96 \times 2 = 1 + 0.92$;
- 28) $0.92 \times 2 = 1 + 0.84$;
- 29) $0.84 \times 2 = 1 + 0.68$;
- 30) $0.68 \times 2 = 1 + 0.36$;
- 31) $0.36 \times 2 = 0 + 0.72$;
- 32) $0.72 \times 2 = 1 + 0.44$;
- 33) $0.44 \times 2 = 0 + 0.88$;
- 34) $0.88 \times 2 = 1 + 0.76$;
- 35) $0.76 \times 2 = 1 + 0.52$;
- 36) $0.52 \times 2 = 1 + 0.04$;
- 37) $0.04 \times 2 = 0 + 0.08$;
- 38) $0.08 \times 2 = 0 + 0.16$;
- 39) $0.16 \times 2 = 0 + 0.32$;
- 40) $0.32 \times 2 = 0 + 0.64$;
- 41) $0.64 \times 2 = 1 + 0.28$;
- 42) $0.28 \times 2 = 0 + 0.56$;
- 43) $0.56 \times 2 = 1 + 0.12$;
- 44) $0.12 \times 2 = 0 + 0.24$;
- 45) $0.24 \times 2 = 0 + 0.48$;
- 46) $0.48 \times 2 = 0 + 0.96$;
- 47) $0.96 \times 2 = 1 + 0.92$;
- 48) $0.92 \times 2 = 1 + 0.84$;
- 49) $0.84 \times 2 = 1 + 0.68$;
- 50) $0.68 \times 2 = 1 + 0.36$;
- 51) $0.36 \times 2 = 0 + 0.72$;

- 52) $0.72 \times 2 = 1 + 0.44$;
- 53) $0.44 \times 2 = 0 + 0.88$;

We did not get any fractional part that was equal to zero. But we had enough iterations (over Mantissa limit) and at least one integer that was different from zero => FULL STOP (losing precision...)

Step 4. Construct the base 2 representation of the fractional part of the number.
Take all the integer parts of the multiplying operations, starting from the top of the constructed list above:

$$0.14_{(10)} =$$
$$0.0010\ 0011\ 1101\ 0111\ 0000\ 1010\ 0011\ 1101\ 0111\ 0000\ 1010\ 0011\ 1101\ 0_{(2)}$$

Step 5. Positive number before normalization:

$$3.14_{(10)} =$$
$$11.0010\ 0011\ 1101\ 0111\ 0000\ 1010\ 0011\ 1101\ 0111\ 0000\ 1010\ 0011\ 1101\ 0_{(2)}$$

Step 6. Normalize the binary representation of the number.
Shift the decimal mark 1 positions to the left so that only one nonzero digit remains to the left of it:

$$3.14_{(10)} =$$
$$11.0010\ 0011\ 1101\ 0111\ 0000\ 1010\ 0011\ 1101\ 0111\ 0000\ 1010\ 0011\ 1101\ 0_{(2)} =$$
$$11.0010\ 0011\ 1101\ 0111\ 0000\ 1010\ 0011\ 1101\ 0111\ 0000\ 1010\ 0011\ 1101\ 0_{(2)} \times 2^0 =$$
$$1.1001\ 0001\ 1110\ 1011\ 1000\ 0101\ 0001\ 1110\ 1011\ 1000\ 0101\ 0001\ 1110\ 10_{(2)} \times 2^1$$

Step 7. Up to this moment, there are the following elements that would feed into the 64-bit double precision IEEE 754 binary floating-point representation:

Sign: 0 (a positive number)
Exponent (unadjusted): 1
Mantissa (not normalized):
1.1001 0001 1110 1011 1000 0101 0001 1110 1011 1000 0101 0001 1110 10

Step 8. Adjust the exponent. Use the 11-bit excess/bias notation:

$$\text{Exponent (adjusted)} =$$
$$\text{Exponent (unadjusted)} + 2^{(11-1)} - 1 =$$
$$1 + 2^{(11-1)} - 1 =$$
$$(1 + 1\ 023)_{(10)} = 1\ 024_{(10)}$$

Step 9. Convert the adjusted exponent from the decimal (base 10) to 11 bit binary.
Use the same technique of repeatedly dividing by 2:

- division = quotient + remainder;
- $1\ 024 \div 2 = 512 + 0$;
- $512 \div 2 = 256 + 0$;
- $256 \div 2 = 128 + 0$;

- 128 ÷ 2 = 64 + 0;
- 64 ÷ 2 = 32 + 0;
- 32 ÷ 2 = 16 + 0;
- 16 ÷ 2 = 8 + 0;
- 8 ÷ 2 = 4 + 0;
- 4 ÷ 2 = 2 + 0;
- 2 ÷ 2 = 1 + 0;
- 1 ÷ 2 = 0 + 1;

Step 10. Construct the base 2 representation of the adjusted exponent.
Take all the remainders starting from the bottom of the list constructed above:
Exponent (adjusted) =

$$1024_{(10)} = 100\ 0000\ 0000_{(2)}$$

Step 11. Normalize the mantissa.

a) Remove the leading (the leftmost) bit, since it's always 1, and the decimal point, if the case.
b) Adjust its length to 52 bits, by removing the excess bits, from the right (if any of the excess bits is set on 1, we are losing precision...).

Mantissa (normalized) =

1. 1001 0001 1110 1011 1000 0101 0001 1110 1011 1000 0101 0001 1110 10 =

1001 0001 1110 1011 1000 0101 0001 1110 1011 1000 0101 0001 1110

Step 12. The three elements that make up the number's 64-bit double precision IEEE 754 binary floating-point representation:

Sign (1 bit) = 0 (a positive number)

Exponent (11 bits) = 100 0000 0000

Mantissa (52 bits) = 1001 0001 1110 1011 1000 0101 0001 1110 1011 1000 0101 0001 1110

Number 3.14 converted from decimal system (base 10) to 64-bit double precision IEEE 754 binary floating point:

0 - 100 0000 0000 - 1001 0001 1110 1011 1000 0101 0001 1110 1011 1000 0101 0001 1110.

When presented in Hex, it would be *40091EB851EB851E*.

3. CONCLUSION

IEEE 754 standard for representation of floating-point numbers provides a simple and logical approach to storing doubles in computer memory. It may be tedious and laborious to convert the representation manually but makes it a lot simpler and solves a lot of problems scientists and engineers faced before the implementation of IEEE 754.

4. REFERENCES

[1] "IEEE Arithmetic." *IEEE Arithmetic Model*, Oracle, 5 Apr. 2000, docs.oracle.com/cd/E19957-01/806-3568/ncg_math.html. (Date Accessed: May 23, 2021)

[2] "IEEE 754." *Wikipedia*, Wikimedia Foundation, 5 Apr. 2021, www.en.wikipedia.org/wiki/IEEE_754. (Date Accessed: May 23, 2021)