laisiotou1997@gmail.com

Why writing this program

When I looked at the 32 bit float variable in memory, I wondering why computer store the floating point number so differently. For example 728.21 is 0x44360d71 in memory, therefore I decided to learn how to do the conversion following IEEE-754.

A strange rounding problem

When I try to normalize 728.21 with my function, the out put will be 728.209961 (**0x44360d70)**, the correct answer for IEEE-754 is **0x44360d71**. The problem is that there are also a rounding algorithm after the mantissa being shift (from 32 bit right shift to 23bit). If the highest bit is 1 according to the dropped bit, 1 will be added to the 23 bit mantissa. Therefore the answer from the function may not be accurate sometime. (Because I did not add that rounding algorithm into the function yet.)

Why people saying float or double is not accurate?

For my understand, it's because of the way that IEEE-754 handling infinite mantissa number and the limitation of 32bit memory size.

When the fractional number is ending with 0.1, the mantissa will be an infinite loop 00110011001100110011001100110011.... As IEEE-754 use low 0bit – 22bit as mantissa, the computer must drop the rest of those bit in order to accomplish that rule(Exponent will also affect the drop bit of mantissa). During the computer demoralize the value in memory it needs to sum 2^x according to the exponent.

2^0		2^-1	2^-2	2^-3	2^-4	2^-5	2^-6	2^-7	2^-8...
1	.	0	0	1	1	0	0	1	1
1	.	+0	+0	+0.125	+0.625	+0	+0	+0.0078125	+0.000390625

As the bit was dropped during the mantissa being shift before, the answer will not be the exact value you enter, unless you somehow saved the dropped bit and add those when denormalize. However it is impossible if the mantissa is infinite loop. Therefore the only value you can get is the closest sum value of that infinite loop mantissa, for example 0.100000001490116119384765625… the more bit, the more accurate for the float number.

Q: Why 0.1 is an infinite mantissa?

1*2
2...0
4...0
8...1
6...1
2...0
4...0
8...1 *Therefore mantissa = 00110011001100110011...*

Last update: 11/05/2020