Assignment-1

Course : E0 251

Author: Rohan G. Shah

M.Tech AI

SR No. : 19243

1. Binary representation(IEEE 754 single precision) of both starting to differ at 19th position starting from left.
2. 22/7 = 010000000100100100**1**0010010010010
3. 0x40490FDB = 010000000100100100**0**0111111011011

For,

X=22/7=3.1428570747375488281250 written in binary form as

X=11.0010010010010010010010=1.100100100**1**0010010010010 \* 2^1

So exponent=1+127(bias)=128,sign=0 and mantissa=10010010010010010010010

For,

Y=0x40490FDB =3.1415927410125732421875

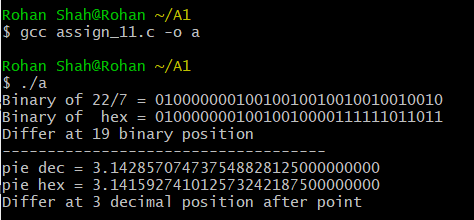
Now we can convert it to simple binary form as

Y= 11.0010010000111111011011 =1.100100100**0**0111111011011 \* 2^1

So in this Binary representation both are differ at 10 th position after binary point(‘.’).

Decimal Representation of both starting to differ at 3rd position after the decimal point(‘.’).

1. 22/7 = 3.14**2**8570747375488281250
2. 0x40490FDB = 3.14**1**5927410125732421875

Output of C program : 

2) IEEE Single precision has

| Sign (1 bit) | Exponent (8 bit) | Mantissa (23 bit) |
| --- | --- | --- |

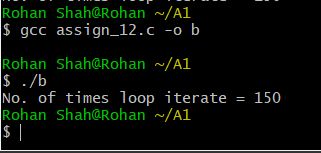
For value 1.0 represented as,

| 0 | 011111111 | 00000000000000000000000 |
| --- | --- | --- |

So when we divide it by 2 iteratively Exponents decrease by value 1 each time till Exponent becomes 0(Exponent is 127 so 127 times loop run before Exponent goes to 0). After that, the floating point value considered as denormalized as Exponent is 0.

So again we divide by 2, we get 1 at the left most bit of Mantissa as 1, after each successive iteration Mantissa values each time right shift by 1 when dividing by 2.

So Total no. of times loops iterate = 127 + 23 = 150

Output of C program : 

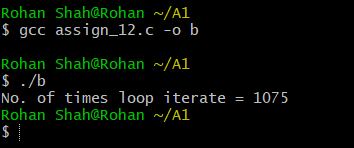
IEEE Double precision has

| Sign (1 bit) | Exponent (11 bit) | Mantissa (52 bit) |
| --- | --- | --- |

value 1.0 represented as,

| 0 | 011111111111 | Zero 52 times |
| --- | --- | --- |

Similar way as in single precision floating point, Total no. of times loops iterate = 1023(bias=2^10-1) + 52(#bits in mantissa)= 1075

Output of C program : 

3) As seen from the output of the C program for n=5 , n=100 etc, the forward and backward harmonic sum is different in fraction part and also backward harmonic sum is often more accurate(checked with different values and compared with online high precision calculator ).

As the forward sum progresses, the sum increases as each subsequent term to be included in the sum decreases.

The greater the difference in magnitude between the two operands of a floating point sum, the more significance is lost from the smaller when added to the larger in the process of aligning the decimal point.(large float number+small float number)

starting with some term, that term and all subsequent terms will cease to contribute to the sum. This can be explained by considering a decimal point shift of the smaller term so extreme that there is no overlap between the significant digits of the larger term and the significant digits of the smaller term, thus the smaller term cannot contribute to the sum. The backward sum, however, maintains a much better between the operands of each binary sum because the magnitude of the successive terms are growing along with the running partial sum.(small float number + slightly big float number compare to operand 1)

The forward sum stops growing at some point, while backward some approximate well.So we get different result in both case.(A+B != B+A in IEEE 754 floating point representation)

Output of C program : 