NYU Tandon School of Engineering

Fall 2021, ECE 6913

Homework Assignment 3

Instructor: Azeez Bhavnagarwala, email: ajb20@nyu.edu

Course Assistant Office Hour Schedule (Room 808, 370 Jay St: 9AM – 11AM)

Mondays & *Tuesdays*: Haotian (Kenny) Zheng **hz2687@nyu.edu** & Shan Hao **sh6206@nyu.edu,**

Wednesdays: Karan Parikh **kap9580@nyu.edu**

Thursday: Sahil Chitnis **ssc9983@nyu.edu**

Fridays: Kewal Jani **kj2062@nyu.edu**

Saturday: Zhiming Fan **zf2035@nyu.edu**

[released Sunday September 26th 2021] [due* **Sunday October 3rd 2021, *before* 11:55 PM**]

You *are allowed* to discuss HW assignments only with other colleagues taking the class. You are *not allowed* to share your solutions with other colleagues in the class. Please feel free to reach out to the Instructor during office hours or by appointment if you need any help with the HW. Please enter your responses in this Word document after you download it from NYU Classes. *Please use the NYU Classes portal to upload your completed HW. Please do not upload images of handwritten sheets or PDFs of scanned sheets of handwritten solutions. Please be sure to type-in your solutions into Word or Google Docs and upload machine readable documents only.*

1.How would you test for overflow, the result of an addition of two 8-bit operands if the operands were (i) unsigned (ii) signed with 2s complement representation.

Add the following 8-bit strings assuming they are (i) *unsigned* (ii) *signed and represented using 2's complement*. Indicate *which of these additions overflow*.

A. 0110 1110 + 1001 1111

B. 1111 1111 + 0000 0001

C. 1000 0000 + 0111 1111

D. 0111 0001 + 0000 1111

A:

(i)0110 1110 + 1001 1111 = 1 0000 1101 Overflow occurs.

(ii)0110 1110 + 1001 1111 = 1 0000 1101 $(13)_{10}$ Not overflow.

B:

(i) 1111 1111 + 0000 0001 = 1 0000 0000 Overflow occurs.

(ii) 1111 1111 + 0000 0001 = 1 0000 0000 $(0)_{10}$ Not overflow.

C:

(i) 1000 0000 + 0111 1111 = 1111 1111 Not overflow

(ii) 1000 0000 + 0111 1111 = 1111 1111 $(-1)_{10}$ Not overflow

D:

(i)0111 0001 + 0000 1111 = 1000 0000 Not overflow

(ii)0111 0001 + 0000 1111 = 1000 0000 $(-128)_{10}$ Overflow occurs

2.One possible performance enhancement is to do a shift and add instead of an actual multiplication. Since 9×6, for example, can be written (2×2×2+1)×6, we can calculate 9×6 by shifting 6 to the left three times and then adding 6 to that result. Show the best way to calculate $0xAB_{hex}$ × $0xEF_{hex}$ using shifts and adds/subtracts. Assume both inputs are 8-bit unsigned integers.

0xAB=16*10+11=171
0xEF=14*16+15=239=2^8-16-1
1010 1011 8times ->1010 1011 0000 0000
1010 1011 4times->1010 1011 0000
Thus we can shift 0xAB to the left 8 times and subtract the result of 0xAB to the left 4 times and finally subtract 0xAB.

3.What decimal number does the 32-bit pattern 0×DEADBEEF represent if it is a floating-point number? Use the IEEE 754 standard

0xDEADBEEF=1 10111101 01011011010111011101111

Sign =1 expoent=1+4+8+16+32+128=189
E-bias= 189-127=62 significand=1.01011011010111011101111= 1.3573893308639526

Thus 0xDEADBEEF=-1.3573893308639526 * 2^{62} = 6.259853398708 x 1018

4. Write down the binary representation of the decimal number `78.75` assuming the `IEEE 754` *single precision* format. Write down the binary representation of the decimal number `78.75` assuming the `IEEE 754` *double precision* format

78.75 = (1001110.11)2

single precision format:

S=0

E-bias=6 E=133 = 1000 0101

Significand=0011 1011

Thus 0 10000101 00111011000000000000000=0x429D8000

double precision format

S=0

E-bias=6 E=1029 = 100 0000 0101

Significand=0011 1011

Thus :

0 10000000101 0011101100000000000000000000000000000000000000000000

=0x4053B00000000000

5.Write down the binary representation of the decimal number `78.75` assuming it was stored using the single precision **IBM format** (base 16, instead of base 2, with 7 bits of exponent).

78.75=0x4E.C=0.4EC*16^2

All HFP formats have 7 bits of exponent with a bias of 64

E-bias=2 E=2+64=66=100 0010

0 1000010 010011101100000000000000

6.`IEEE 754-2008` contains a half precision that is only `16 bits` wide. The leftmost bit is still the sign bit, the exponent is `5 bits` wide and has a `bias of 15`, and the mantissa (fractional field) is `10 bits` long. A hidden 1 is assumed.

(a) Write down the bit pattern to represent **-1.3625×10^{-1}** Comment on how the range and accuracy of this 16-bit floating point format compares to the single precision `IEEE 754` standard.

(b) Calculate the sum of **1.6125×10^{1}** `(A)` and **3.150390625** $\times 10^{-1}$ `(B)` by hand, assuming operands A and B are stored in the 16- bit half precision described in problem a. above Assume `1 guard, 1 round bit, and 1 sticky bit`, and round to the nearest even. Show all the steps.

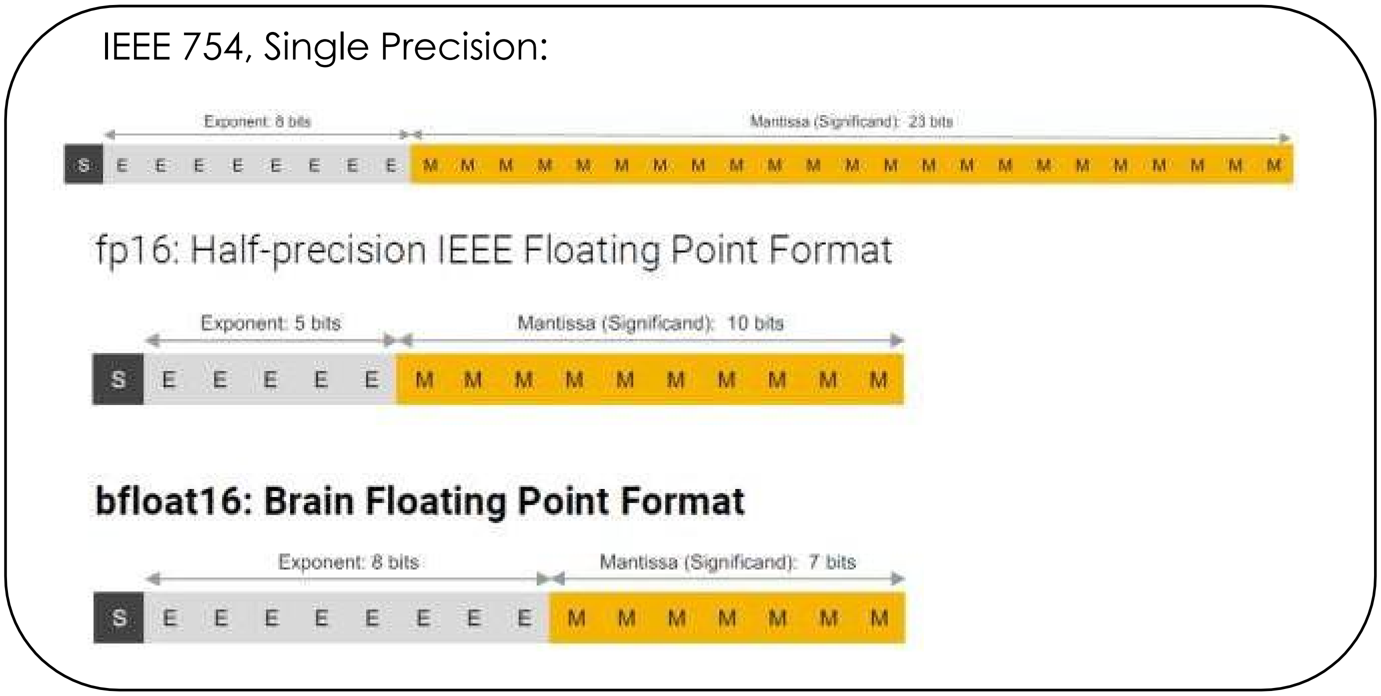
(a)
-1.3625×10^{-1}=-0.13628=-1.0001011100*2^{-3}
Sign=1
E-bias=-3 E=12=01100
1011000001011100
Thus compared to the single precision IEEE 754 standard, it is not precisive.

(b)
1.6125*10^{1}=16.125=10000.001=1.0000001*2^{4}
Sign=0
E-bias=4 E=19=1 0011
0100110000001000
Guard=1 round=0 sticky=1
3.150390625*10^{-1}=0.010100001010011001100=1.0100001010011001100*2^{-2}
Sign=0
E-bias=-2 E=13=0 1101
0011010100001010

1.0000001000*2^{4}+0.0000010100*2^{4}=1.0000011100*2^{4}=16.4375

7.What is the range of representation and relative accuracy of positive numbers forthe following 3 formats:

(i) `IEEE 754` Single Precision (ii) `IEEE 754 - 2008`(described in Problem 6 above) and (iii) ‘`bfloat16`’ shown in the figure below


IEEE 754, Single Precision:
Exponent: 8 bits
Mantissa (Significand): 23 bits
S E E E E E E E E M M M M M M M M M M M M M M M M M M M M M M M
fp16: Half-precision IEEE Floating Point Format
Exponent: 5 bits
Mantissa (Significand): 10 bits
S E E E E E M M M M M M M M M M
bfloat16: Brain Floating Point Format
Exponent: 8 bits
Mantissa (Significand): 7 bits
S E E E E E E E E M M M M M M M

(i) $(-2*2^{127},-1*2^{-126}] \cup [2^{-126},2*2^{127})$
accuracy =2^{-23}

(ii)$(-2*2^{15},-1*2^{-14}] \cup [2^{-14},2*2^{15})$
accuracy =2^{-10}

(iii)$(-2*2^{127},-1*2^{-126}]\cup[2^{-126},2*2^{127})$
accuracy $=2^{-7}$

8.Suppose we have a 7-bit computer that uses IEEE floating-point arithmetic where a floating point number has 1 sign bit, 3 exponent bits, and 3 fraction bits. All of the bits in the hardware work properly.

Recall that denormalized numbers will have an exponent of 000, and the `bias` for a 3-bit exponent is

$2^{3-1} - 1 = 3$.

(a) For each of the following, write the *binary value* and *the corresponding decimal value* of the 7-bit floating point number that is the closest available representation of the requested number. If rounding is necessary use round-to-nearest. Give the decimal values either as whole numbers or fractions. The first few lines are filled in for you.

Number	Binary	Decimal
0	0 000 000	0.0
-0.125	1 000 000	-0.125
Smallest positive normalized number	0001000	0.25
largest positive normalized number	0110111	15
Smallest positive denormalized number > 0	0000001	0.03125
largest positive denormalized number > 0	0000111	0.21875

(b) The associative law for addition says that a + (b + c) = (a + b) + c. This holds for regular arithmetic, but does not always hold for floating-point numbers. Using the 7-bit floating-point system described above, give an example of three floating-point numbers a, b, and c for which the associative law does not hold, and show why the law does not hold for those three numbers.

When adding two numbers will result overflow, the associative law can't be used.

For example a=01111111 b=0111110 c=1111000 a+(b+c) can get the right anwser, but a+b will lead to overflow.