NYU Tandon School of Engineering

Fall 2023, ECE 6913

Homework Assignment 3

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

Course Assistant Office Hour Schedule

On Zoom: 9:30AM – 11AM Monday, Tuesday, Wednesday & Thursday

On Zoom: 4PM – 6PM Monday (Focus on Project A)

1. Arushi Arora, aa10350@nyu.edu
2. Prashanth Rebala, pr2359@nyu.edu
3. Moin Khan, mk8793@nyu.edu
4. Raj Ghodasara, g4357@nyu.edu

Homework Assignment 3 [released Friday October 6th 2022] [due Sunday October 15th by 11:59PM]

You *are allowed* to discuss HW assignments with anyone. You are *not allowed* to share your solutions with other colleagues in the class. Please feel free to reach out to the Course Assistants or 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 Brightspace portal to upload your completed HW.*

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`

(i) unsigned

A. 0110 1110 + 1001 1111 = 1 0000 1101 => overflow

B. 1111 1111 + 0000 0001 = 1 0000 0000 => overflow

C. 1000 0000 + 0111 1111 = 1111 1111 = 255_{10} => no overflow

D. 0111 0001 + 0000 1111 = 1000 0000 = 128_{10} => no overflow

(ii) signed

A. 0110 1110 + 1001 1111 = 0000 1101 = 13_{10} => no overflow

B. 1111 1111 + 0000 0001 = 0000 0000 = 0_{10} => no overflow

C. 1000 0000 + 0111 1111 = 1111 1111 = -1_{10} => no overflow

D. 0111 0001 + 0000 1111 = 1000 0000 => overflow

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 = 1010 1011 = $171_{10} = 2^7 + 2^5 + 2^3 + 2^1 + 2^0$
0xEF = 1110 1111 = 239_{10}

```
1. Shift 0xEF left 7 places.
2. Add 0xEF shifted left 5 places.
3. Add 0xEF shifted left 3 places.
4. Add 0xEF shifted left 1 place.
5. Add 0xEF.
```

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 = 1101 1110 1010 1101 1011 1110 1110 1111

sign bit = 1
Exp bits = $10111101_2 = 189_{10}$
Frac bits = 01011011011111011101111
Value of exponent = E - 127 = 62
Value in decimal = $-(1.01011011011111011101111)_2 \times 2^{62}$
= $-6.259853398708 \times 10^{18}$

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

$$78.75_{10} = 1001110.11_2 = 1.00111011_2 \times 2^6$$

sign = 0

fraction = 0011 1011

exponent = 6+Bias

Single: 6+127=133=1000 0101_2

Double: 6+1023=1029=100 0000 0101_2

Single: 0100 0010 1001 1101 1000 0000 0000 0000

Double: 0100 0000 0101 0011 1011 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

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).

$$78.75_{10} = 1001110.11_2 = 0.0100111011_2 \times 16^2$$

sign = 0

fraction = 0100 1110 1100 0000 0000 0000

exponent = 2+Bias=2+64=66=100 0010_2

IBM format: 0100 0010 0100 1110 1100 0000 0000 0000

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.

$$-1.3625 \times 10^{-1} = -0.0010001100_2 = -1.00011 \times 2^{-3}$$

sign = 1
fraction = 0.0001100000
exp = -3+bias=-3+15=12=01100_2
the expression is 1011 0000 0110 0000

(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.

$$1.6125 \times 10^1 = 16.125 = 10000.001 = 1.0000001 \times 2^4$$
$$3.150390625 \times 10^{-1} = 0.3150390625 = 0.010100001010 = 1.0100001010 \times 2^{-2}$$
$$= 0.0000010100001010 \times 2^4$$

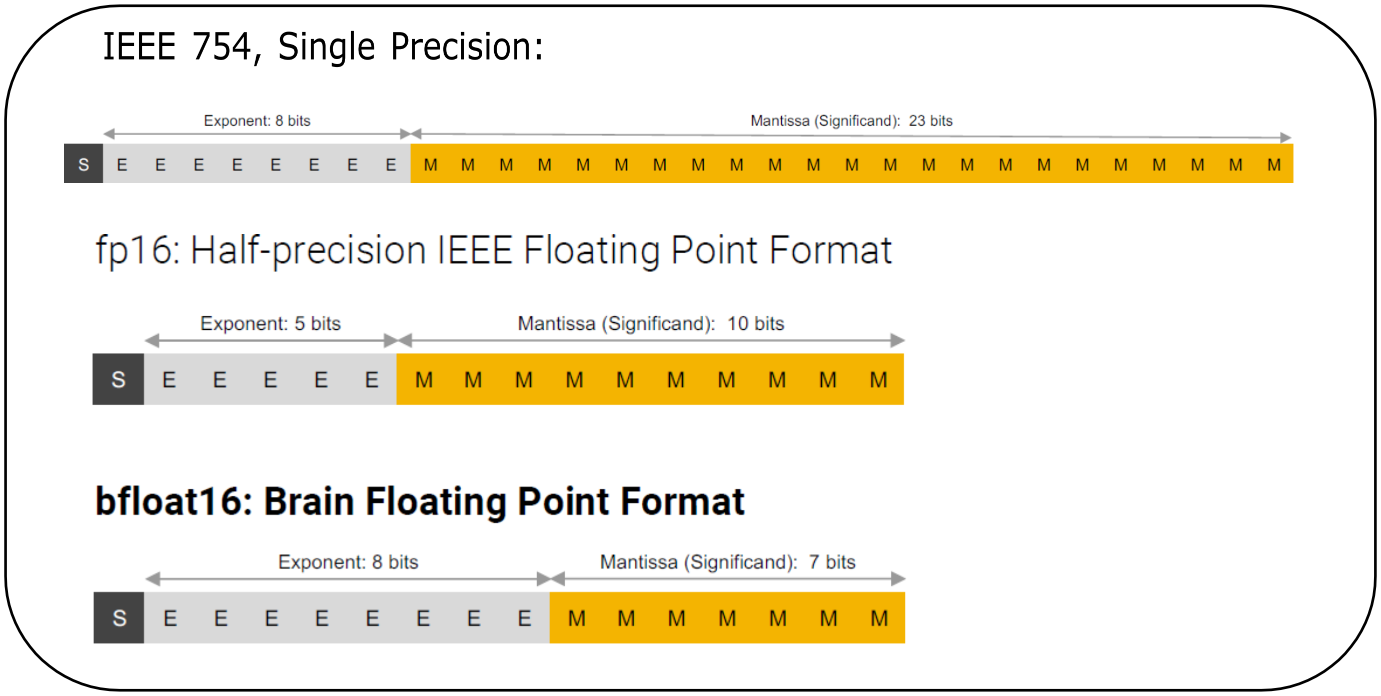
1.0000001000 00
0.0000010100 00 1010

1.0000011100 00

$1.000001110000 \times 2^4 = 10000.011100 \times 2^0 = 16.375$

7. What is the range of representation and relative accuracy of positive numbers for the 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

IEEE 754 SP has the same number of bits in the Exponent Field (8) as bfloat16 and thus has a comparable range (marginally higher range).

Both IEEE 754 SP and bfloat16 have a larger range than fp16 or Half-precision IEEE which has only 5 bits for the exponential field.

IEEE 754 SP has a higher precision than both fp16 and bfloat16 since it has a larger fractional field of 23 bits compared to fp16 (10 bits) or bfloat16 (7 bits).

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	0 001 000	0.25
largest positive normalized number	0 110 111	15
Smallest positive denormalized number > 0	0 000 001	0.03125
largest positive denormalized number > 0	0 000 111	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.

a=1 110 111

b=0 110 111

c=0 000 001

Then (a + b) + c = c, because a and b cancel each other, while a + (b + c) = 0, because b + c = b (c is very small relative to b and is lost in the addition).