NYU Tandon School of Engineering

Fall 2021, ECE 6913

Homework Assignment 3 Solutions

---

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*.

(i) unsigned addition

A. 0110 1110 + 1001 1111 = 1 0000 1101 [overflow since there is carry over with the MSB]

B. 1111 1111 + 0000 0001 = 1 0000 0000 [overflow since there is carry over with the MSB]

C. 1000 0000 + 0111 1111 = 1111 1111 (=255)

D. 0111 0001 + 0000 1111 = 1000 0000 (=128)

(ii) signed addition *and represented using 2's complement.*

A. 0110 1110 + 1001 1111 = 0000 1101; [110] + [-97] = [+13] **No overflow**

B. 1111 1111 + 0000 0001 = 1111 1111 +1 = 0000 0000; [-1] + [1] = 0-bit string result of addition of operands is '0' [carry over from MSB is discarded since this is signed addition] **No overflow**

C. 1000 0000 + 0111 1111 = 1111 1111; [-128] + [127] = -1 ; **No overflow**

D. 0111 0001 + 0000 1111 = 1111 1111; [113] + [15] = +128 Here, the sign bit of the result is different from the sign bit of the operands. **Overflow has occurred.** +128 is outside the range of 2s complement representation using 8 bits

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 $(2 \times 2 \times 2 + 1) \times 6$, 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 $0xAB_{hex} \times 0xEF_{hex}$ using shifts and adds/subtracts. Assume both inputs are 8-bit unsigned integers.

$0xAB_{16}$ = 1010 1011 = 171_{10}
$0xEF_{16}$ = 1110 1111 = 239_{10}

$171_{10} \times 239_{10} = 40869_{10} =$ **$9FA5_{16}$**

$0xAB_{16} \times 0xEF_{16} = = 57121_{10} = 0xDF21_{16}$
$0xAB_{16} = 10101011_2 = 171_{10}$, and $171 = 128 + 32 + 8 + 2 + 1$

$= 2^7 + 2^5 + 2^3 + 2^1 + 2^0$

We can shift 0xEF left **7 places** = 111 0111 1000 0000 = 7780_{16}
then add 0xEF shifted left **5 places** = 1 1101 1110 0000 = $1DE0_{16}$
then add 0xEF shifted left **3 places** = 0111 0111 1000 = 778_{16}
then add 0xEF shifted **once** 0000 1 1101 1110 = $1DE_{16}$
and then add 0xEF = 1110 1111 = EF_{16}
7780 + 1DE0 + 778 + 1DE + EF = **$0x9FA5_{16}$.**
5 shifts, 4 adds.

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

$DEADBEEF_{16} = 3735928559_{10}$

in binary as a 32-bit string:

1101 1110 1010 1101 1011 1110 1110 1111

in IEEE 754 single precision:

1 10111101 01011011011111011101111

Exponent field = 1011 1101 = BD = 189_{10}

Value of exponent = E − 127 = 62

$1.01011011011111011101111 \times 2^{62}$

$6.259853398708 \times 10^{18}$

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 × 10^0 = 1001110.11 × 2^0
To normalize, move binary point *six places* to the *left:*
1.00111011 × 2^6
sign = positive, exp = 127 + 6 = 133
Final bit pattern: 0 1000 0101 0011 1011 0000 0000 0000 000
= 0100 0010 1001 1101 1000 0000 0000 0000 = 0x429D8000

The above solution is the representation for single precision format.

The *double precision format* is as below:

78.75 × 10^0 = 1001110.11 × 2^0
To normalize, move binary point *six places* to the *left*:

1.00111011 × 2^6
sign = positive, exp = 1023 + 6 = 1029_{10} = 100 0000 0101_2

Final bit pattern:

= 0 100 0000 0101 0011 1011 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 = 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_{10} in decimal, equal to **0100 1110.1100 0000_2** in binary and when represented in hex **= $4E.C0_{16}$**
we normalize the hex by shifting right 1 hex digit (4 bits) at a time until the leftmost digit is 0: **0.4EC0 x 16^2**

Exponent Bias in IBM format = 64
Exponent – Bias = val(E) = 2

So, exponent = bias + 2 = 64+2 = 66
66 = 1000010
Final bit pattern
0 1000 010 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.

$-1.3625 \times 10^{-1} = -0.13625 \times 10^{0}$
$= -0.0010001100 \times 2^{0}$
To normalize, move the binary point three to the right= -1.00011×2^{-3}
Sign bit = 1
fraction = 0.0001100000

Bias for N=5 bit exponent field = $2^{N-1}-1 = 15$
value of exponent = exponent field – Bias, so, – 3 = exponent field - Bias = exp field – 15

so, exp field = -3 + 15 = 12

16 bit representation is thus: 1 01100 0001100000

(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.

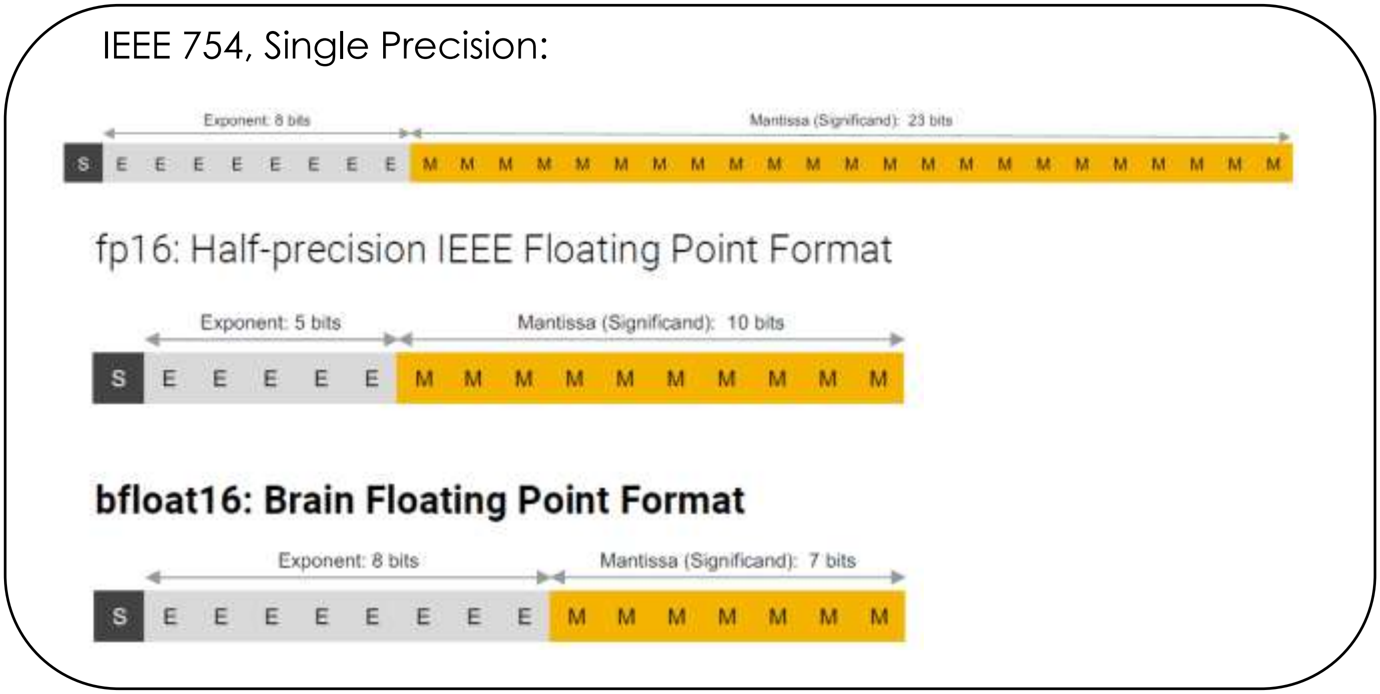
$1.6125 \times 10^{1} + 3.150390625 \times 10^{-1}$
$1.6125 \times 10^{1} = 16.125 = 10000.001 =$ **1.0000001×2^{4}**
$3.150390625 \times 10^{-1} = 0.3150390625 = 0.010100001010 =$ **$1.0100001010 \times 2^{-2}$**
Shift binary point six places to the left to align exponents,
$0.0000010100001010 \times 2^{4}$

```
               GR
1.0000001000 00
0.0000010100 00 1010 (Guard 0, Round 0, Sticky 1)
-------------------
1.0000011100 00
--------------------
```

In this case both GR bits are 0
Thus, the value is same with stick bit discarded.
$1.0000011100 \times 2^{4} = 10000.011100 \times 2^{0} = 16.4375$

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 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		
largest positive normalized number		
Smallest positive denormalized number > 0		
largest positive denormalized number > 0		

First row:
The number 0 is represented by all 0s in the Exponent field and all 0s in the Fraction field.
Second row: -0.125: $N = -0.125$ in decimal corresponds to -0.001_2 in binary [since $2^{-3} = 0.125$] Normalizing -0.001_2 by shifting the binary point right 3 places, we get, $N = -1.000_2 \times 2^{-3}$ Given, 1 sign bit, 3 exponent field bits, 3 fraction field bits and bias of 3,
Sign bit = 1 (Exponent – bias) = value of exponent in normalized form (Exponent – 3) = value of exponent = -3 [since N in normalized form = $-1.000_2 \times 2^{-3}$] So, Exponent = 0 i.e., exponent field = 000
Fraction field = 000 [since N in normalized form = -1.0002×2^{-3}]
So, Binary representation with S E F fields is: 1 000 000

Third row: **smallest possible normalized number**:
Exponent field: all zeros except LSB: 001; Fraction field: all 0s, sign bit = 0
S E F: 0 001 000
value of exponent = E – 3 = 1-3 = -2
Fraction field = 0.000
so, value of representation = 1.00×2^{-2}
= 0.25

Fifth row: **Smallest possible denormalized number:**
S = 0; E=000; F = 001; leading digit of significand = 0
Denormalized numbers have an exponent that is all 0s, implying an exponent equal to
1-bias, or 1-3, or -2 in this case.

(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.

There are several possible answers. Here's one:

Let *a* = 1 110 111, *b* = 0 110 111, and *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).