CSE 321b: Computer Organization (II)
Third Year, Computer & Systems Engineering

Solution to Assignment #3

1. Apply **Booth's** algorithm to multiply -12 (multiplicand) by +6 (multiplier). Represent the numbers using the least number of bits.

A	Q	Q_{-1}	M	
00000	00110	0	10100	Initial values
00000	00011	0		Shift
01100	00011	0		A ← A – M
00110	00001	1		Shift
00011	00000	1		Shift
10111	00000	1		A ← A + M
11011	10000	0		Shift
11101	11000	0		Shift

2. Show all the steps required to divide +14 (dividend) by -5 (divisor) using the **non-restoring division** algorithm. Represent the numbers using the least number of bits.

A	Q	M	
00000	01110	11011	Initial values
00000	01110	00101	Take absolute of Q & M
00000	1110?		Shift
11011			Subtract
11011	1110**0**		Q_0 ← 0
10111	1100?		Shift
11100			Add
11100	1100**0**		Q_0 ← 0
11001	1000?		Shift
11110			Add
11110	1000**0**		Q_0 ← 0
11101	0000?		Shift
00010			Add
00010	0000**1**		Q_0 ← 1
00100	0001?		Shift
11111			Subtract
11111	0001**0**		Q_0 ← 0
00100	00010		Add
00100	11110		Adjust signs of A & Q

3. Consider the IEEE 754 **half-precision** format (which is also known as: **binary16**) in which floating point numbers are represented using 16 bits: 1 sign bit, 5-bit biased exponent, and 10-bit fraction. Convert the following numbers to their IEEE half-precision counterparts:
(a) $4.57763671875*10^{-5}$

$4.57763671875*10^{-5} = 2^{-14.405} = 0.75 * 2^{-14}$ ➔ 0 00000 1100000000

(b) -217.375

$-217.375 = -2^{7.764} = -1.6982421875 * 2^{7}$ ➔ 1 10110 1011001011

4. Suppose the IEEE 754 Standard has a **binary14** format that uses: 1 sign bit, 6-bit biased exponent, and 7-bit fraction. Perform the following calculations while interpreting each of the given binary values as a binary16 floating-point number. Use two guard bits and round the result to the **nearest** binary16 number whenever is necessary.

   (a) 1 101001 1110000 + 0 100111 1000010

   i) Check for special cases
      ➔ No special cases
   ii) Transform subtraction to addition and negate second number
      ➔ Not needed
   iii) Align
      ➔ Second number has a smaller exponent
      ➔ Add 2 to its exponent and shift its fraction to the right twice
      ➔ Exponent of second number = 101001
      ➔ Significand of second number = 0.011000010
   iv) Add significands (taking signs into consideration)
      ➔ Significand of result = -1.111000000 + 0.011000010 = -1.011111110 (no overflow!)
      ➔ Fraction of result = 011111110
      ➔ Sign bit of result = 1
   v) Normalize
      ➔ Not needed
      ➔ Exponent of result = 101001
   vi) Round
      ➔ Candidate fractions are 0111111 and 1000000
      ➔ Guard bits = 10
      ➔ Round to the even candidate
      ➔ Fraction of result = 1000000

   Result = 1 101001 1000000

   (b) 1 111111 0000000 × 1 000000 0000000

   i) Check for special cases
      ➔ First number represents -∞, Second number represents -0
      ➔ Result is qNaN

   Result = x 11111 1xxxxx

   (c) 1 011101 0000001 ÷ 0 100010 1111101

   i) Check for special cases
      ➔ No special cases
   ii) Subtract exponents (and add the bias)

- ➔ Exponent of result = 011101 – 100010 + 011111 = 011010 (no overflow/underflow!)

iii) Calculate sign of result
- ➔ Numbers have different signs
- ➔ Sign bit of result = 1

iv) Divide significands

```
                                0 . 1 0 0 0 0 0 1 0 1
1 1 1 1 1 1 0 1 | 1 0 0 0 0 0 0 1 . 0 0 0 0 0 0 0 0 0
                    1 1 1 1 1 1 0   1
                              1 0   1 0 0 0 0 0 0
                                1   1 1 1 1 1 0 1
                                    1 0 0 0 0 1 1 0 0
                                      1 1 1 1 1 1 0 1
                                              1 1 1 1
```

- ➔ Significand of result = 0.100000101

v) Normalize
- ➔ Subtract 1 from exponent and shift significant (one position to the) left
- ➔ Exponent of result = 011001 (no underflow)
- ➔ Fraction of result = 000001010

vi) Round
- ➔ Candidate fractions are 0000010 and 0000011
- ➔ Guard bits = 10
- ➔ Round to the even candidate
- ➔ Fraction of result = 0000010

Result = 1 011001 0000010