COEN 020 Name: ______________________________ Fall 2017

Quiz #8

Monday, November 13th

1. [10 pts] Complete the implementation of the function `CircleArea` that calculates and returns the area of a circle (πr^2) using Q32 fixed-point reals. Your solution may call the library functions `Q32Ratio` and `Q32Product` as needed, but do not write those functions.

```
// Data type declaration:
typedef int64_t Q32 ;

// Prototypes of library functions:
Q32 Q32Ratio(int32_t top, int32_t btm) ;
Q32 Q32Product(Q32 a, Q32 b) ;
```

```
Q32 CircleArea(Q32 r)
      {
      Q32 pi = Q32Ratio(314159, 100000) ;
      Q32 rSquared = Q32Product(r, r) ;
      return Q32Product(pi, rSquared) ;
      }
```

Most did reasonably well on this problem. However, you can't initialize a fixed-point real using a floating-point constant, as in Q32 pi = 3.14159. The Q32 data type is really a 64-bit integer, so assigning a real value to an integer causes all fractional data to be discarded. Also, you can't declare the same identifier twice, as in Q32 area … followed by Q32 area … Finally, you can't use the multiplication operator to multiply two Q32 numbers; you have to use the Q32Product

2. [10 pts] The following function uses Q32 fixed-point reals to calculate the area of a triangle. Translate it into ARM assembly language:

```
Q32 TriangleArea(Q32 base, Q32 height)
      {
      return Q32Product(base, height) >> 1 ;
      }
```

```
TriangleArea:
      PUSH   {LR}
      BL     Q32Product
      LSRS   R1,R1,1
      RRX    R0,R0
      POP    {PC}
```

Most wrote a sequence of multiply and shift instructions to compute a Q16 product but the problem uses Q32. Very few called the function Q32Product as indicated in the C code. Since Q32 values use a 64-bit representation, the right shift by 1 bit to divide by 2 must be a 64-bit double-length shift; it cannot be a single shift or divide instruction.

3. [10 pts] Calculate the product $A \times B = 1002_{10} \times 3004_{10}$ by decomposing the operands into their least and most-significant halves. Give the value of each of the four partial products, indicate which operands were used to produce each partial product, show their relative position for summation, and perform that addition to produce the final result. Do all of your work in decimal.

$A_{HI} = 10_{10}$, $A_{LO} = 02_{10}$
$B_{HI} = 30_{10}$, $B_{LO} = 04_{10}$

```
AHIBHI:   0300
AHIBLO:     0040
ALOBHI:     0060
ALOBLO:       0008
          ________
          03010008₁₀
```

The point of the math isn't to determine the result; it's to help understand how to code 64 bits x 64 bits on a 32-bit processor. Decomposition separates two double-length (4 digit) operands into four single-length (2 digit) operands. Decomposing A yields 10 and 02, not 1000 and 02. Each partial product should then be 2 digits × 2 digits → 4 digits.

1. [10 pts] The unsigned product of the two binary integers 1011_2 and 0110_2 is 01000010_2. Show in clear detail the arithmetic operation(s) and the relative positions of their operands that is required to convert the unsigned product into a signed 2's complement product. Do all of your work in binary.

1011 is negative, so subtract 0110 from the most-significant half of 01000010:

```
 0100 0010
-0110
 1110 0010
```

Note: This problem has nothing to do with the algorithm for changing the sign of a 2's complement number. We previously learned that the only difference between signed and unsigned double length products is the upper half of the result. Converting an unsigned to a signed product should therefore require some operation on the upper half without affecting the lower half of the product. What we learned was that if either operand is negative, we subtract the other operand from the most-significant half of the unsigned product.