FLOATING POINT TANGENT AND DIVISION SUBROUTINES

OVERVIEW:
This laboratory has two programming components. The first is to convert your result from the sine function into a floating point number using a special simplified floating point format. For the second part of this lab you will develop a tangent function routine using this floating point sine and cosine format. This will require developing a floating point divide routine. Furthermore, you will need to monitor your division routine for possible numeric overflows during the calculation process.

Details of the Floating Point Conversion:
Your output, i.e. F(x), must be converted to a special form called floating point to be used by other programs. A "floating point" representation of a number writes the number as a mantissa and a signed exponent according to

```
        x=mantissa*2^exponent
```

where * denotes multiplication and the mantissa is restricted to the range [0.5,1]. This means that the mantissa is always greater than or equal to 1/2 but less than 1. To help you with understanding this concept consider the representation of x=3 in our notation.

```
        x = 3 = %11.000 = %0.11 * 2^(+2) = $0.C000 * 2^(+2)
```

Similarily,

```
        x = -3  =  $-0.C000 * 2^(+2)

        x = 3.1 =  $0.C666 * 2^(+2)

        x = 1/8 =  $0.8000 * 2^(-2)

        x = -4096 =  $-0.8000 * 2^(+12) = $-0.C000 *
2^(+0C)
```

(Note that the exponent is now written as $+OC)

```
        x = -4095 =  $-0.FFE0 * 2^(+11) = $-0.C000 *
2^(+0B)
```

(Note that the exponent is now written as $+OB)

Your program should return the sine and/or cosine output from lab#5 in "floating point" format as defined above and store it in memory according to the format shown in Figure 1.

```
    15         bits           0
    ----------------------------
    |        MANTISSA          |    Word #1
    ----------------------------
```

```
|  sign(1)  |  exponent(2) |   Word #2
----------------------------
```

NOTES:
(1) sign is a byte representing the sign of the mantissa;
(2) exponent is a signed (2's complement) byte representing the exponent.

Figure 1 - EEAP 282 Floating Point Number Representation

Note that this is a two word format. The sign need be only a single bit but, for convenience, we have defined it to be a byte. To help you understand this format several examples are shown in Figure 2.

```
        --------------
X=3     |   $C000    |
        --------------
        | $00 | $02  |
        --------------


        --------------
X=3.1   |   $C666    |
        --------------
        | $00 | $02  |
        --------------


        --------------
X=-3    |   $C000    |
        --------------
        | $FF | $02  |
        --------------


        --------------
X=1/8   |    $8000   |
        -------------
        | $00 | $-02 |
        --------------


        --------------
X=-4096 |    $8000   |
        -------------
        | $FF | $0D  |
        --------------


        --------------
X=-4095 |    $FFF0   |
        -------------
        | $FF | $0C  |
        --------------
```

Figure 2 - Examples of the EEAP 282 floating point number format

Although details of the final writeup will be forthcoming, these questions will be expected to be addressed in your lab report.

1. What are the limits of our floating point notation?

2. How can the tangent of all angles be represented using our notation. Pay proper attention to angles near 0°, 90°, 180°, 270° and 360°.

Here are some sample results for the floating point sine, cosine and tangent functions. There is some ambiguity about zeros which you might want to address in your write-up. For example, should a zero be positive or negative. Most commercial Floating Point Units (FPUs) will accept either either a positive zero or a negative zero. Where this is possible is shown below as $00 or $FF being ok for the sign byte when the mantissa is $0000.

Angle	Fixed SIN	Floating SIN	Fixed COS	Floating COS	Floating TAN
03E8	0620	C4 00 00 FD	3FB4	FE D0 00 00	C4 E9 00 FD
3330	3CC5	F3 14 00 00	1413	A0 98 00 FF	C1 B2 00 02
5BA0	31D7	C7 5C 00 00	A824	A0 90 FF 00	9E ED FF 01
6000	2D41	B5 04 00 00	AD41	B5 04 FF 00	80 00 FF 01
C350	BFC8	FF 20 FF 00	0532	A6 40 00 00	C4 6D FF 04
4000	4000	80 00 00 01	0000/8000	00 00 FF 00 00	??? / 0
8000	0000/8000	00 00 FF 00 00	C000	80 00 FF 01	00 00 00 00 FF
C000	C000	80 00 FF 01	0000	00 00 00 00 FF	??? / 0

```
                      ------ -- --
mantissa <- - |    |  |--> exponent
                   |
                  sign
```

This is the format of our answers. Note that exponent is a 2's complement byte.

Some detailed floating point calculations:

```
Angle = $03E8

SIN($03E8) = $0620 = %00 00 0110 0010 0000
                        ^
                      implied binary point in format
```

Converting to floating point format by shifting

```
                   = +% 1100 0110 0000 0000 x 2^-3
                       ^
                      implied binary point in format
                   = +$0.C400 x 2^-3

COS($03E8) = $3FB4 = %00 11 1111 1011 0100
                        ^
                      implied binary point in format
```

Converting to floating point format by shifting

```
                   = +% 1111 1110 1101 0000 x 2^0
                       ^
                      implied binary point in format
                   = +$0.FED0 x 2^0

TAN($03E8) = SIN($03E8)/COS($03E8)
           = (+$0.C400 x 2^-3)/(+$0.FED0 x 2^0)
           = +$0.C4E9 x 2^-3
```

NOTE: You will typically have to convert this division to `$C4000000/$FED0` for the DIVU instruction to give a non-zero quotient. This should be addressed in your lab report.

```
Angle = $3300

SIN($3300) = $3CC5 = %00 11 1100 1100 0101
                        ^
                      implied binary point in format
```

Converting to floating point format by shifting

```
                   = +% 1111 0011 0001 0100 x 2^0
                       ^
                      implied binary point in format
                   = +$0.F314 x 2^0

COS($3300) = $1413 = %00 01 0100 0001 0011
                        ^
                      implied binary point in format
```

Converting to floating point format by shifting

```
                   = +% 1010 0000 1001 1000 x 2^-1
                       ^
                      implied binary point in format
                   = +$0.A098 x 2^-1

TAN($3300) = SIN($3300)/COS($3300)
           = (+$0.F314 x 2^0)/(+$0.A098 x 2^-1)
           = +$1.837C x 2^1
```

Note that this answer cannnot be produced by the DIVU instruction unless you shift the floating point before the DIVU instruction. The reason is that the leading 1 makes the number larger than the word length quotient of the DIVU instruction. Renormalizing,

```
TAN($3300) = +$1.837C x 2^1
           = +%0001 1000 0011 0111 1100 x2^1
                   ^
               implied binary point
```

after shifting

```
           = +% 1100 0001 1011 1100 x 2^2
               ^implied binary point
           = +$0.C1BE x 2^2
```

```
Angle = $5BA0

SIN($5BA0) = $31D7 = %00 11 0001 1101 0111
                        ^
                      implied binary point in format
```

Converting to floating point format by shifting

```
                   = +% 1100 0111 0101 1100 x 2^0
                       ^
                      implied binary point in format
                   = +$0.C75C x 2^0

COS($5BA0) = $A824 = %10 10 1000 0010 0100
                        ^
                      implied binary point in format
```

Converting to floating point format by shifting

```
                   = -% 1010 0000 1001 0000 x 2^0
                       ^
                      implied binary point in format
                   = -$0.A090 x 2^0
```

```
TAN($5BA0) = SIN($5BA0)/COS($5BA0)
           = (+$0.C75C x 2^0)/(-$0.A090 x 2^0)
           = -$1.3DDB x 2^0
```

Renormalizing,

```
TAN($5BA0) = -$1.3DDB x 2^0
           = -%0001 0011 1101 1101 1011 x2^1
                   ^
               implied binary point
```

after shifting

```
           = +% 1001 1110 1110 1101 x 2^2
               ^implied binary point
           = +$0.9EED x 2^2
```

```
Angle = $6000

SIN($6000) = $2D41 = %00 10 1101 0100 0001
                        ^
                      implied binary point in format
```

Converting to floating point format by shifting

```
                   = +% 1011 0101 0000 0100 x 2^0
                       ^
                      implied binary point in format
                   = +$0.B504 x 2^0

COS($6000) = $AD41 = %10 10 1101 0100 0001
                        ^
                      implied binary point in format
```

Converting to floating point format by shifting

```
                   = -% 1011 0101 0000 0100 x 2^0
                       ^
                      implied binary point in format
                   = -$0.B504 x 2^0

TAN($6000) = SIN($6000)/COS($6000)
           = (+$0.B504 x 2^0)/(-$0.B504 x 2^0)
           = -$1.0000 x 2^0
```

Renormalizing,

```
TAN($6000) = -$1.0000 x 2^0
           = -%0001 0000 0000 0000 0000 x2^1
                   ^
               implied binary point
```

after shifting

```
            = -% 1000 0000 0000 0000 x 2^1
                ^implied binary point
            = -$0.8000 x 2^1
```

```
Angle = $C350

SIN($C350) = $BFC8 = %10 11 1111 1100 1000
                           ^
                      implied binary point in format
```

Converting to floating point format by shifting

```
                    = -% 1111 1111 0010 0000 x 2^0
                          ^
                      implied binary point in format
                    = -$0.FF20 x 2^0
```

```
COS($C350) = $0532 = %00 00 0101 0011 0010
                           ^
                      implied binary point in format
```

Converting to floating point format by shifting

```
                    = +% 1010 0110 0110 0000 x 2^-3
                          ^
                      implied binary point in format
                    = +$0.A660 x 2^-3
```

```
TAN($C350) = SIN($C350)/COS($C350)
           = (-$0.FF20 x 2^0)/(+$0.A660 x 2^-3)
           = -$1.888F x 2^3
```

Renormalizing,

```
TAN($C350) = -$1.888F x 2^3
           = -%0001 1000 1000 1000 1111 x2^3
                   ^
               implied binary point
```

after shifting

```
           = -% 1100 0100 0100 0111 x 2^4
               ^implied binary point
           = -$0.C447 x 2^4
```

Extra Credit Lab #1

This lab will not be checked out. You must submit a formal lab report consisting of:

1. title page in the normal format
2. pseudocode description of the program. This should not be simply the comments from your assembly language code.
3. assembler listing of the program
4. table of results for 8 randomly chosed angles PLUS 180 and 270 degrees. This table should list the digital angle, the floating point sine, the floating point cosine, AND the floating point tangent values.

This should be accompanied by a short explanation of what the program does.
You should be specific about

1. how you handle mathematical overflow in the tangent calculations. Specifically, what is the largest number you can represent with this floating point scheme and how have you chosen to handle the issue of the tangent for 90 degrees. Simply avoiding dividing by zero is not enough.
2. how you implemented the floating point division for the mantissa. Did you re-normalize the result after division? What instruction(s) did you use for the divide?