# TRIG FUNCTIONS

OVERVIEW: This laboratory will implement an interpolation routine to compute the sine and cosine of an input binary number. Input will be from a data register. You will use an interpolation table to compute the sine of the angle; the cosine can be computed from the cosine since cos(x)=-sin(x-90 degrees). You will then display both results to the debugger screen.

CHECKOUT: There will be a scheduled checkout for this lab beginning November 13th in the Kern Lab. Schedules will be e-mailed to you.

EXTRA CREDIT: Extra credit equivalent to 20% of the lab can be obtained by implementing a scheme to input your angle from the keyboard and control your program in calculating the sine and cosine, i.e. make your program similar to a trig function calculator. The amount of extra credit will depend upon your implementation. This extra credit project must be turned in by November 10th.

BACKGROUND: Trigonometric functions are NOT part of microprocessor instruction sets; however, they are among the most useful functions for real world engineering applications such as robotic control. For the purposes of this lab we may imagine a robotic elbow which is free to rotate 360 degrees. A sensor, mounted on the robot elbow, measures the angle as a 16-bit binary number. This means that each 360 degree rotation of the elbow is divided into 65536 (2 to the 16th power) discrete angular increments. The details of this angular numbering scheme are shown in Figure 1 where x is the digitally encoded angle:

```
                 $4000=16384=90 degrees
                              |
                              |
             QUADRANT II      |    QUADRANT I
                              |
                              |     /
                              |    /
                              |   /angle x
  32768 =                     |  /
180 degrees __________________|/______________ 0 = 0 degrees
  =$8000                      |        $10000=65536=360
degrees
                              |
                              |
             QUADRANT III     |    QUADRANT IV
                              |
                              |
                              |
                  $C000=49152=270 degrees
```

Figure 1 - Definition of angular measurements (Angular representation of axes are shown)

Note that according to Figure 1 each angular increment is equal to 360 degrees/65,536 = 5.5x10-3 degrees = 9.587x10-5 radians. The sensor's actual input to the computer is a 16- bit word; however, you do not have 65,536 words of memory to store these sine values nor do you know a fast algorithm to compute the sine function. So you are clever. The sine function values repeat themselves every 90 degrees so you need only store values for angles between 0 and 90 degrees. The sine for angles between 90 degrees and 360 degrees can be simply computed using such trig relationships as sin(phi)=sin(90 degrees-phi), etc. Storing the values for the sine of angles between 0 and 90 degrees still requires 16,384 values and is more memory than you have available. So you decide to store only 64 values of the sine function, i.e. you will store a value corresponding to every 90 degrees/64=1.41 degrees, and interpolate to find any intermediate values. Using this scheme and our previous definition of 65,636 angles we can now code a 16-bit angle according to the format shown in Figure 2.

```
____________________________________________________________
__|   |   |   |   |   |   |   |   |   |   |   |   |   |   |
1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
1
__|___|___|___|___|___|___|___|___|___|___|___|___|___|___|
__

quadrant
<---->|<--------index-------->|<------lowest 8 bits--------
->
        one of 64 known values   must interpolate this part
              in table           if not equal to 00000000

              x[13:8]                   x-x[16:8]
```

Figure 2 - Encoding of a 16 bit binary angle

Interpretation of Figure 2:

1. The two most significant bits indicate the quadrant of the angle. Since there are four quadrants (1, 2, 3, and 4 in decimal); their binary equivalent is 00, 01, 10 and 11. The quadrant will determine the specific trig relationship we need to compute the sine and the sign of the resulting function.
2. The next six bits represent the 64 evenly spaced binary angles in that quadrant, i.e. every 1.41 degrees. 3. The last eight bits indicate exactly where the angle lies between two known 1.41 degrees increments and will be used for interpolation. For example, if the last eight binary bits are 00000000 the angle is exactly one of the 64 known angles whereas if the eight binary bits are 10000000 the angle lies exactly half-way between two known angles.

PROBLEM STATEMENT:

PART 1 - The Sine Function: The sine of a 16-bit angle (as described in Figure 2) will be computed from a table of 64 known evenly spaced angular values and interpolated (as necessary) to get an accurate value of sine(x). Assuming your input angle is x in the format of Figure 2, you will estimate sine(x) according to the following formula:

sine(x) = f(x[13:8]) + (x-x[16:8])*D(x[13:8]) {1}

where x[13:8] is the six bits of x corresponding to the 64 known values of sine x in a table in your program, f( ) represents the actual value of one of the 64 known sine values, D( ) is the derivative of the sine function at the index value x[13:8]. (x-x[16:8]) is a formal way of saying using just the lower eight bits of x; it says take the original 16 bit angle and subtract from it the upper eight bits leaving all zeros in the upper byte of x and the original bits in the lower byte. Equation {1} is a digital equivalent of the well known Taylor Series approximation, Equation {2}.

f(x) = f(xi) + (x-xi)*df(xi)/dx {2}

Note that (x-xi) is the fractional part of the angle between two of the 64 known angles,xi and xi+1, and thus must be the least significant byte of the input word x. If you know f(xi) and df(xi)/dx you can estimate sine(x) in the first quadrant. By examining the quadrant of x you can approximate sine(x) for x>90 degrees. However, remember that the sine is negative in quadrants III and IV.

The values of f(xi) and D(xi) [actually f(x[13:8]) and D(x[13;8])] were supplied to you as tables of numbers which you typed into the computer for Lab #1. Instructions for including them in your program appear in Appendix IV. These tables are arranged in order of increasing angle as shown in Appendix III.

Note that angles which are multiples of 90 degrees such as 180 degrees or 270 degrees are not included in TBL or DTBL. The easiest way to handle these angles is to test for them and treat each case separately. Other methods such as extending the TBL and DTBL tables are also possible and left to your discretion; however, you must document your method of dealing with these angles in your lab report.

NOTE: Differences and derivatives are the same in the content of this problem. What you are really doing is optimized linear interpolation when you implement Equation (2).

Programming Considerations:

ASSEMBLING AND LINKING YOUR PROGRAM: The data tables of known sine values should be typed in by you as a separate program module. You will combine them with your program in memory using the LINKER. Details of doing this are given in Appendix IV.

Your data table program module must begin with the following instruction XDEF TBL,DTBL and end with the normal END. The XDEF command must be before the ORG statements and will tell the assembler and linker that these names are global (i.e., they can be used by other program modules). Your lab 4 program must then begin with the instruction XREF TBL,DTBL. This tells the assembler that these names are defined elsewhere. Note that these instructions should appear before any include io.s statement you may use. If you look at your symbol table after you assemble the lab you will see the word EXTERNAL indicating that the linker must find the reference for these names. Finally, to put the programs together you must use a slightly different linker command, i.e. something of the form ld68k -o lab4 lab4 data or ld68k -L -o lab4 lab4 data > lab4.llis if you want to get a listing of the linker output. The linker output really doesn't tell you much unless you have a very complicated program. Note that the data is in a file called data.s in this example.

Your program is now fairly complex. You may have i/o routines defined at $9000, your data table might be at $2000, and you are putting your program somewhere else. Just be careful that you don't put the program on top of the data table or something similar. The data tables shown in Appendix II begin at $2000 but you don't have to put them there. The value of memory used here ($2000) was arbitrarily chosen and you are free to change the assignment if you want. For example, you can put your program at $2000 and TBL beginning at $3000.

ACCESSING THE DATA: You can access individual elements (which are one word in length) of the tables in the following manner:

```
LEA       TBL,A0      ;put pointer to TBL in A0
MOVE      #2,D0       ;put the number of the element you
                      ;want to retrieve in D0. Note that 2
                      is only used as an example. You should
                      generate this number from the binary
                      angle in your program.
LSL       #2,D0       ;multiply D0 x 4 to account for zeros
                      between elements
MOVE.W    (A0,D0),D1  ;get the (D0/2+1)-th element from
                      TBL and put it in D1
```

What you are putting into D0 is your offset (i.e. element) in TBL. D0 is defined by you and must be in the range of 0-63 since the table only has 64 elements in it. The offset corresponding to the element you want is doubled to allow the above program fragment to automatically skip over the zero words stored between each actual element in TBL. This extra space was reserved for additional data which might appear in a later lab. For this lab you will only want to access the evenly spaced words in TBL skipping the zeros. The difference table DTBL is in the same format and you can repeat the above using LEA DTBL,A0 to access the data table DTBL.

TBL is the sine function table for 64 quadrant I angles only. Each entry consists of 2 bytes (formatted as one word) followed by a filler word of decimal value zero - there are

128 values in the table of which you will only use 64. The numbers stored in TBL are stored in a signed magnitude format with the binary point between bits 14 and 13, i.e. the format used by TBL is:

bit 15 sign bit: 0 indicates positive number, 1 indicates negative number. Note that all the elements in this table are positive so this bit is always zero.
bit 14 most significant bit. A TBL entry of 0100 0000 0000 0000 would correspond to a value of +1, i.e. the sine of 90 degrees. Note that the value of the sine for an angle of 90 degrees is NOT included in the table. YOU have to decide how to handle the sine of 90 degrees. Possible options include extending TBL and DTBL to include 90 degrees or to specifically test for angles that are multiples of 90 degrees.

As an aside the elements in TBL were calculated by the formula
TBL(n)=sin(n*90 degrees/64)*16384
which gives the decimal entries shown in Appendix II for TBL. For example, if n=63
TBL(63)=sin(63*90 degrees/64)*16384 = 16379.07 which is (after roundoff) exactly the entry shown in Appendix II.

According to our previous discussion DTBL is a table of derivatives corresponding to the elements of TBL and also has 128 values. In actual fact, DTBL is a special table constructed to approximate the sine function by linear interpolation with minimum error. The details of this scheme can be found in "Fast Trig Functions for Robot Control" by Carl Ruoff in the November/December 1981 issue of Robotics Age magazine. Basically, DTBL was constructed by the formula

DTBL(n)=[sin((n+1)*90 degrees/64)-sin((n)*90 degrees/64)+delta(n)]*16384

where delta(n) is a small constant that was added to the difference to improve the average accuracy of the final interpolation. The details of this are discussed in the article by Ruoff. A basic problem of the above formula is that the differences are very small numbers and, if we used the same numerical format to represent these differences as that used in TBL, all the entries in DTBL would be zeros except possibly for the least significant 2 or 3 bits. To keep more significant digits in the interpolation calculation I shifted the binary points of the numbers generated by the above formula 13 bits to the left, i.e. I multiplied the DTBL* calculations by 8192 to get the final DTBL entries:

Because of this additional multiplication by 8192 you will have to divide your interpolation by 8192 BEFORE adding it to the entry in TBL. Specifically, you will multiply (x-x[16:8]) by D(x[13:8]) using a MUL command to generate a long word interpolation according to Equation (1) but you cannot simply add this result to f(x[13:8]) as stored in TBL. The result of (x-x[16:8])*D(x[13:8]) MUST be shifted 13 bits to the right (divided by 8192!) before adding to align the binary points for addition. NOTE: If you used a MULU instruction your multiplication result will be a long word and a 13-bit shift will put what you want to add to f(x[13:8]) in the lower word of your answer.

Since you may not be sure of your answers the following correct answers are provided for you to test your program:

| ANGLE | INDEX | FRACTION | INTERPOLATED | SINE | QUADRANT | 03E8 |
|---|---|---|---|---|---|---|
| 03 | E8 | 016A | 0620 | I | | |
| 3300 | 33 | 00 | 0000 | 3CC5 | I | |
| 5BA0 | 24 | 60 | 005E | 31D7 | II | |
| C350 | 3C | B0 | 0017 | BFC8 | IV | |
| A000 | 20 | 00 | 0000 | AD41 | III | |
| 4000* | 00 | 00 | 0000 | 4000 | | |

*90 degrees

NOTE: All answers are in hex.

Detailed calculation of these results is done in Appendix I.

You can use these formulas to calculate the sine in all four quadrants:

| | |
|---|---|
| quadrant 1 | sine(x) |
| quadrant 2 | sine(180 degrees - x) |
| quadrant 3 | -sine(x- 180 degrees) |
| quadrant 4 | -sine(360 degrees - x) |

Other formulas may be faster.

## APPENDIX I SAMPLE SINE CALCULATIONS

EXAMPLE 1: x=$03E8=00 000011 11101000 The breaks in the binary number show the three parts of the digital angle which we can refer to as quadrant, index and fraction.

The first two bits tell us this is a first quadrant angle which can be directly interpolated using Equation (1).

The next six bits have value $03 which is the index of this angle relative to the 64 angles (i.e. 0 to 63).

The next eight bits have the value $E8 and represent the fractional angle between $03 and $04 and will be used for interpolation.

Looking up the relevant parameters from TBL and DTBL:

```
TBL($03) = $04B5
DTBL($03)= $3214
```

Performing the multiplication:

```
(x-x[16:8]) * D(x[13:8]) = $E8 * $3214 = $002D6220
```

Shift this result 13 bits to the right by dividing by 8192 give

```
$016B
```

Performing the final addition:

```
Sine($03E8) = $04B5 + $016B = $0620
```

EXAMPLE 2: x=$3300=00 110011 00000000 The first two bits tell us this is a first quadrant angle which can be directly interpolated using Equation (1).

The next six bits have value $33 which is the index of this angle relative to the 64 angles (i.e. 0 to 63).

The next eight bits have the value $00 and tell us no interpolation is necessary.

Looking up the relevant parameters from TBL and DTBL:

```
        TBL($33) = $3CC5
        DTBL($33)= $0F2E
```

Performing the multiplication:

```
        (x-x[16:8]) * D(x[13:8]) = $00 * $0F2E = $00000000
```

Shift this result 13 bits to the right by dividing by 8192 gives

```
        $0000
```

Performing the final addition:

```
        Sine($3300) = $3CC5 + $0000 = $3CC5
```

EXAMPLE 3: x=$A000=10 100000 00000000 The first two bits tell us this is a third quadrant angle which will require computing the complement of the angle, i.e. x=$A000-$8000=$2000=00 100000 00000000 The value $8000 is the hex equivalent of 32,768 or 180 degrees in Figure 1. This angle is now a first quadrant angle and can be computed as we have done the previous examples.

The next six bits now have value $20 which is the index of this angle relative to the 64 angles (i.e. 0 to 63).

The next eight bits have the value $00 and tell us no interpolation is required.

Looking up the relevant parameters from TBL and DTBL:

```
        TBL($20) = $2D41
        DTBL($20)= $231A
```

Performing the multiplication:

```
        (x-x[16:8]) * D(x[13:8]) = $00 * $231A = $00000000
```

Shift this result 13 bits to the right by dividing by 8192 gives

```
        $0000
```

Performing the final addition:

```
        Sine($2000) = $2D41 + $0000 = $2D41
```

Since this is a third quadrant angle bit 15 will be set to 1 to represent a negative number, or

```
        Sine($A000) = -Sine($2000) = $AD41
```

EXAMPLE 4: x=$5BA0=01 011011 10100000 The first two bits tell us this is a second quadrant angle which will require computing the complement of the angle, i.e. x=$8000-$5BA0=$2460=00 100100 01100000 The value $8000 is the hex equivalent of 32,768 or 180 degrees in our original notation. This angle is now a first quadrant angle and can be computed as we have done the previous examples.

The new next six bits have value $24 which is the index of this angle relative to the 64 angles (i.e. 0 to 63).

The next eight bits have the value $60 and tell us that interpolation is required.

Looking up the relevant parameters from TBL and DTBL:

```
          TBL($24) = $3179  or  TBL(36)=12665
          DTBL($24)= $1F68  or DTBL(36)=8040
```

Performing the multiplication:

```
          (x-x[16:8]) * D(x[13:8]) = $60 * $1F68 = $0006C700
```

Shift this result 13 bits to the right by dividing by 8192 gives

```
          $005E
```

Performing the final addition:

```
          Sine($2460) = $3179 + $005E = $31D7
```

Since this is a SECOND quadrant angle bit 15 remains zero, or

```
          Sine($5BA0) = Sine($2460) = $31D7
```

EXAMPLE 5: x=$C350=11 000011 10100000 The first two bits tell us this is a FOURTH quadrant angle which will require computing the complement of the angle, i.e. x=$10000-$C350=$3CB0=00 111100 10110000 The value $10000 is the hex equivalent of 65,536 or 360 degrees in our original notation. This angle is now a first quadrant angle and can be computed as we have done in the previous examples.

The new next six bits have value $3C which is the index of this angle relative to the 64 angles (i.e. 0 to 63).

The next eight bits have the value $B0 and tell us interpolation is required.

Looking up the relevant parameters from TBL and DTBL:

```
          TBL($3C) = $3FB1
          DTBL($3C)= $0450
```

Performing the multiplication:

```
          (x-x[16:8]) * D(x[13:8]) = $B0 * $0450 = $0002F700
```

Shift this result 13 bits to the right by dividing by 8192 gives

```
          $0017
```

Performing the final addition:

```
          Sine($3CB0) = $3FB1 + $0017 = $3CB0
```

Since this is a FOURTH quadrant angle, the sine is negative and bit 15 is set to 1, giving

```
          Sine($C350) = -Sine($3CB0) = $BFC8
```

## APPENDIX II

Sample data.s program

```
XDEF    TBL,DTBL
```

```
     ORG   $2000
TBL  DC.W  0,0,402,0,804,0,1205,0,1606,0,2006,0,2404,0
     DC.W  2801,0,3196,0,3590,0,3981,0,4370,0,4756,0
     DC.W  5139,0,5520,0,5897,0,6270,0,6639,0,7005,0,7366,0
     DC.W  7723,0,8076,0,8423,0,8765,0,9102,0,9434,0,9760,0
     DC.W  10080,0,10394,0,10702,0,11003,0,11297,0,11585,0
     DC.W  11866,0,12140,0,12406,0,12665,0,12916,0,13160,0
     DC.W  13395,0,13623,0,13842,0,14053,0,14256,0,14449,0
     DC.W  14635,0,14811,0,14978,0,15137,0,15286,0,15426,0
     DC.W  15557,0,15679,0,15791,0,15893,0,15986,0,16069,0
     DC.W  16143,0,16207,0,16261,0,16305,0,16340,0,16364,0
     DC.W  16379,0
DTBL DC.W  12867,0,12859,0,12843,0,12820,0,12789,0,12751,0
     DC.W  12704,0,12650,0,12589,0,12519,0,12443,0,12358,0
     DC.W  12267,0,12168,0,12061,0,11948,0,11827,0,11699,0
     DC.W  11564,0,11422,0,11273,0,11117,0,10955,0,10786,0
     DC.W  10611,0,10429,0,10241,0,10046,0,9846,0,9640,0
     DC.W  9427,0,9210,0,8986,0,8758,0,8524,0,8285,0
     DC.W  8040,0,7792,0,7538,0,7280,0,7017,0,6750,0
     DC.W  6479,0,6205,0,5926,0,5644,0,5359,0,5070,0
     DC.W  4778,0,4483,0,4186,0,3886,0,3584,0,3280,0
     DC.W  2973,0,2665,0,2356,0,2044,0,1732,0,1419,0
     DC.W  1104,0,789,0,474,0,158,0
```

## APPENDIX III MATHEMATICAL ORGANIZATION OF TBL AND DTBL

```
      Memory Location       Contents        Digital angle (x)
TBL   $2000               sine(0 degrees)     0000000000000000
      $2002               0
      $2004               sine(1.41 degrees)  0000000100000000
      $2006               0
      $2008               sine(2.82 degrees)  0000001000000000
      $200A               0
      $200C               sine(4.23 degrees)  0000001100000000
      $200E               0
        .                 .                   .
        .                 .                   .
        .                 .                   .
      $20FC               sine(88.59 degrees) 0011111100000000
      $20FE               0

DTBL  $2100               d{sine(x)}/dx@x=0 degrees
      $2102               0
      $2104               d{sine(x)}/dx@x=1.41 degrees
      $2106               0
      $2108               d{sine(x)}/dx@x=2.82 degrees
      $210A               0
        .                 .
        .                 .
        .                 .
      $21FC               d{sine(x)}/dx@x=88.59 degrees
      $21FE               0
```

Note that the angles x corresponding to the values of sine and d{sine(x)}dx are not shown for DTBL since they are the same, i.e. the entry in TBL at $2000 and the entry in DTBL at $2100 correspond to x=0000000000000000. The entries in TBL at $2004 and $2104 correspond to x=0000000100000000, etc.

## APPENDIX IV HOW TO LINK YOUR DATA PROGRAM MODULE WITH YOUR LAB#4 PROGRAM USING THE LINKER

```
<data.s file>
     XDEF TBL,DTBL
ORG  $2000
TBL  DC.W   0,0,402,0,804,0,1205,0,1606,0,2006,0,2404,0
     DC.W   2801,0,3196,0,3590,0,3981,0,4370,0,4756,0
     DC.W
5139,0,5520,0,5897,0,6270,0,6639,0,7005,0,7366,0
     DC.W
7723,0,8076,0,8423,0,8765,0,9102,0,9434,0,9760,0
     etc

<lab4.s file>
     XREF   TBL,DTBL
        include io.s
     ...REST OF YOUR PROGRAM...
```

The above XDEF and XREF statements MUST be put into your files using the vi editor. Then, the files must be linked together using a link command of the form:

```
lnk68k -L -o lab4 data,lab4 >lab4.llis
```

Note that data and lab4 are indicated as input files; lab4 is also designated as the name of the output file. For those of you that are interested:

XDEF is an assembler directive which tells the assembler that the symbol names following it are being defined for use by other program modules (i.e. files).

XREF is an assembler directive the defines symbols in a program module (file) as being defined in another program module (file).

The purpose of the XREF and XDEF directives is to allow you to construct an assembly language program that is spread across several files. This typically makes debugging the program easier and allows the programmer to develop an efficient program design.

Subject: Lab #4 Write-up

First. You should not compute the cosine of the angle. That will be the subject of the next lab. Second, many people have asked about the proper form of program i/o. Your program can accept an input from a register loaded with the debugger or, alternatively from a DC.W statement in your program. The choice is up to you. I would not recommend using either HexIn or data input through $10040/$10042 because that will use up the monitor window. You should similarily keep your output simple - a value in a register or memory location would be the accepted form of output.

Your Lab #4 report should follow the following format:
Standard Title Page with the legal statement that you did the enclosed work.

Description of your program. This description should consist of
1. pseudo code or a flow chart description of your program
2. complete description of how to input and output data from your program. Be specific as to where the inputs and outputs are and the format they are in. 3. a complete description of memory requirements (i.e. how many bytes long is your program and where is it in memory), and a listing of all registers and memory locations used by your program and a short description of what each register is used for
4. a copy of your program listing (your .lis file, not your .s file)
5. An explanation of how this lab works. This is an essay type question and should be no longer than a page. It can be much shorter but it should address the idea of table look up and interpolation. A discussion of how accurate the sine routine is should be included. Hint: compute the sine for a given angle using your program and compare your results with the results from a calculator. This write-up should describe the Taylor series computation approach, the fact that DTBL is not strictly speaking the exact derivative at that point, and any other factors that influence the calculation. This explanation should specifically address the following:
    - how you looked up date from your program
    - how the interpolation was accomplished in this program
    - why you have to use a LSR #13 after the multiplication in your program
6. table of results

YOU MUST INCLUDE A TABLE OF TWELVE RANDOMLY CHOSEN ANGLES FOR WHICH YOU COMPUTE THE SINE USING YOUR PROGRAM. THIS TABLE MUST INCLUDE ONE ANGLE FROM EACH QUADRANT AND AT LEAST ONE ANGLE WHICH IS A MULTIPLE OF 90 DEGREES AND SHOULD BE OF THE FORM

| ANGLE | SINE |
|---|---|

NOTE errors in this table of sine values will cost you points.

>>  The program crashes when I use an indirect referance to A1
>>(the command where it crashes is something like move.w (A1,D1),angle; where
>>angle is a label I defined to store the value of the indexed angle) my
>>program halts due to exception vector 3, which signifies address error.
>>needless to say I am quite stuck.

I have gotten many messages like the above.  An exception vector 3 error is an address error - an attempt to access a word or long word at an odd address.  This is typically caused by improper usage of the index.  TBL and DTBL are word length.

P.S. For your information a complete list of exception vector error messages is found on the inside back cover of your textbook.

This is a list of angles you can use for checking out Lab #4.  You are not responsible for the cosine yet.

| angle | sine | cosine |
|---|---|---|
| 0010 | 0019 | 3FFF |
| 0100 | 0192 | 3FFB |
| 0330 | 0500 | 3FCD |
| 03E8 | 0620 | 3FB4 |
| 0500 | 07D6 | 3F85 |
| 077F | 0BB5 | 3EEA |
| 1000 | 187E | 3B21 |
| 2000 | 2D41 | 2D41 |
| 3000 | 3B21 | 187E |
| 3300 | 3CC5 | 1413 |
| 3FFF | 3FFF | 0001 |
| 4000 | 0000 | 8000 |
| 5000 | 3B21 | 987E |
| 5B00 | 3274 | A760 |
| 5BA0 | 31D7 | A824 |
| 6000 | 2D41 | AD41 |
| 7000 | 187E | BB21 |
| 8000 | 8000 | 8000 |
| 9000 | 987E | BB21 |
| 9AE0 | A738 | B292 |
| A000 | AD41 | AD41 |
| AD00 | B92B | 9CC6 |
| B000 | BB21 | 987E |
| C000 | 8000 | 0000 |
| C350 | BFC8 | 0532 |
| C4DE | BF8A | 07A0 |
| D000 | BB21 | 187E |
| E000 | AD41 | 2D41 |
| E456 | A830 | 31CD |

```
F000    987E    3B21
```

Students should not have computed the cosine. If they did ignore it for the purposes of grading this lab.

>Your Lab #4 report should follow the following format:
>Standard Title Page with the legal statement that you did the enclosed work.
(5 points)

>Description of your program. This description should consist of
>1. pseudo code or a flow chart description of your program
(10 points, -3 no quadrants, -3 no 90 degree angles)

>2. complete description of how to input and output data from your program.
>Be specific as to where the inputs and outputs are and the format they are
>in.
(5 points)

>3. a complete description of memory requirements (i.e. how many bytes long
>is your program and where is it in memory), and a listing of all registers
>and memory locations used by your program and a short description of what
>each register is used for
(5 points)

>4. a copy of your program listing (your .lis file, not your .s file)
(10 points)

>5. An explanation of how this lab works. This is an essay type question
>and should be no longer than a page. It can be much shorter but it should
>address the idea of table look up and interpolation. A discussion of how
>accurate the sine routine is should be included. Hint: compute the sine
>for a given angle using your program and compare your results with the
>results from a calculator. This write-up should describe the Taylor series
>computation approach, the fact that DTBL is not strictly speaking the exact
>derivative at that point, and any other factors that influence the
>calculation. This explanation should specifically address the following:
>  how you looked up date from your program (10 points)
>  how the interpolation was accomplished in this program (10 points)
>  why you have to use a LSR #13 after the multiplication in your program
(10 points)
>  accuracy of the routine (11 points)
For this last point, they should make some attempt to compare the results for a known angle with the result of their routine. Many of them do not seem to understand the format of the output so we should look for an understanding of the error.

(41 points)

>6. table of results
(24 points, -2 for each wrong answer)
I have a basic program which can compute the correct answers.

>YOU MUST INCLUDE A TABLE OF TWELVE RANDOMLY CHOSEN ANGLES FOR WHICH YOU
>COMPUTE THE SINE USING YOUR PROGRAM.  THIS TABLE MUST INCLUDE ONE ANGLE
>FROM EACH QUADRANT AND AT LEAST ONE ANGLE WHICH IS A MULTIPLE OF 90 DEGREES
>AND SHOULD BE OF THE FORM
>
>ANGLE      SINE
>
>NOTE errors in this table of sine values will cost you points.

The total is then 100 points.