FLOATING POINT TRIGONOMETRIC FUNCTION IMPLEMENTATION

History:

Trigonometric functions are NOT part of microprocessor instruction sets; however, they are among the most useful functions for real world applications such as robotic control. We may imagine a robot elbow which is free to rotate 360 degrees. A sensor (called a rotary encoder) is mounted on the robot elbow which measures the angle in binary. For the particular sensor we will use a 360 degree rotation is divided into 65536 (2^{16}) discrete angular increments, i.e. any anglular position of the robot can be represented as a 16-bit (word) integer. The details of the angular numbering scheme are shown below:

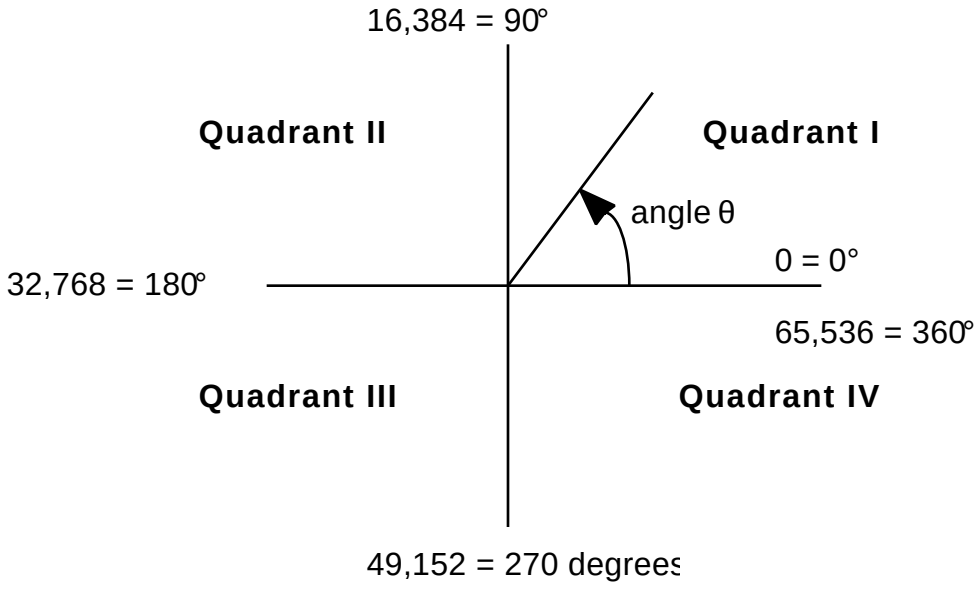
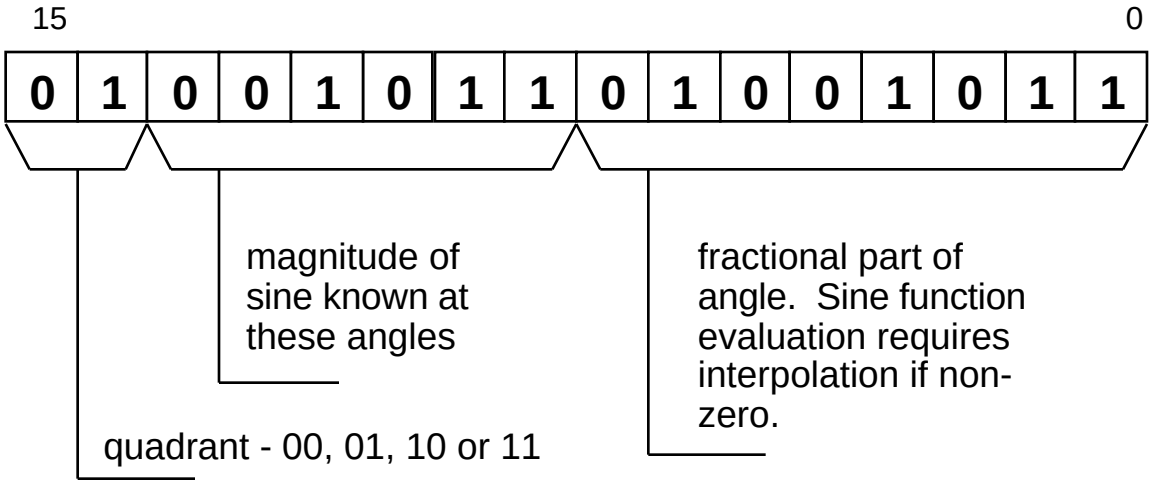


Figure 1 - Definition of angular measurements

Note that according to the above figure each angular increment is equal to 360 degrees/65,536 = 5.5×10^{-3} degrees = 9.587×10^{-5} radians. The sensor's actual input to the computer is a 16-bit word with the following format:



The sine of the input angular word (described above) can be computed using a known table of 64 values and interpolating to get a more accurate answer. Assuming your input angle is x, you will estimate sine(x) as F(x) according to the following formula:

$$F(x) = f(x[13:8]) + (x-x[16:8]) * D(x[13:8])$$

where x[13:8] are 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, and D() is a table of differences between each of

the sine functions. This could be written, in a more conventional fashion, as

$$F(x)=f(x_i) + (x-x_i)\, D(x_i)$$

Note that $(x-x_i)$ is the fractional part of the angle between two known points, x_i and $x_{i+1,}$ and is the least significant byte of the input word x. Remember that the sine is negative in quadrants III and IV.

The values of $f(x_i)$ and $D(x_i)$ are supplied as tables of numbers.

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 writes a number as a mantissa and a signed exponent according to

$$x = \text{mantissa} * 2^{\text{exponent}}$$

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.

$x = 3_{10} = 11.000_2 = 0.11_2 \times 2^{+2} = 0.C000_{16} \times 2^2$

Similarily,

$x = -3_{10} = -0.C000_{16} \times 2^{+2}$

$x = 3.1_{10} = 0.C666_{16} \times 2^{+2}$

$x = 1/8_{10} = 0.8000_{16} \times 2^{-2}$

$x = -4096_{10} = -0.8000_{16} \times 2^{+12} = -0.C000_{16} \times 2^{+0C}$
(Note that the exponent is now written as $+OC_{16}$)

$x = -4095_{10} = -0.FFE0_{16} \times 2^{+11} = -0.C000_{16} \times 2^{+0B}$
(Note that the exponent is now written as $+OB_{16}$)

Your program ahould convert the number x, which is your interpolated sine value, into "floating point" as defined above and store it in memory according to the following format.

15 0

MANTISSA (word)		Word #1
Sign of mantissa (byte)	Signed exponent (byte)	Word #2

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 (corresponding to the previous numberical examples) are shown below.

X=3

$C000	
$00	$02

X=3.1

$C666	
$00	$02

X=-3

$C000	
$FF	$02

X=1/8

$FFE0	
$FF	$0B

X=-4096

$8000	
$FF	$0C

X=-4095

$FFE0	
$FF	$0B

Programming considerations:

The data tables of known sine values will be supplied to you beginning at address $2000. To access these data tables use the following assembler directive at the top of your program.

```
    XREF    TBL, DTBL
```

This tells the assembler that these labels will be defined by another program which will be included by the linker. To include these data tables in your program use the command:

```
ld68k -L -o lab4 data,lab4 >lab4.llis
```

when linking your program. Note that lab4.s contains your program, and data.s contains your sine data. The data tables are stored in memory beginning at $2000 so don't put your program on top of the data tables. Put your program at $1000 or something similarily far away from the data tables. 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        index to TBL in D0,
                      use 0,4,8,etc.
MOVE.W (A0,D0),D1     get the (D0/2+1)-th
                      element of TBL
```

What you put 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 reason for using long word offsets, i.e.4,8, etc. is that a zero word is stored between each actual element in TBL. This extra space is reserved for additional data which might appear in another 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 start of the sine function table for quadrant I 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. DTBL is a table of differences corresponding to the elements of TBL and also has 128 values. However, because of the dynamic range, i.e. smallness of the differences as compared to the values in TBL, you will have to do some

additional math when using the difference table. Specifically, you can multiply (x-x[16:8]) * D(x[13:8]) using a MUL command to generate a long word result. But, the most significant word of this result, which is what will be added to f() must be shifted 13 bits to the right to align the binary points for addition. Be careful here since you must also convert your long word multiplication result to a word before shifting the binary point for correct addition.

Actual Tables of Data:

index	TBL	DTBL
1	0	12867
	0	0
2	402	12859
	0	0
3	804	12843
	0	0
4	1205	12820
	0	0
5	1606	12789
	0	0
6	2006	12751
	0	0
7	2404	12704
	0	0
8	2801	12650
	0	0
9	3196	12589
	0	0
10	3590	12519
	0	0
11	3981	12443
	0	0
12	4370	12358
	0	0
13	4756	12267
	0	0
14	5139	12168
	0	0
15	5520	12061
	0	0
16	5897	11948
	0	0
17	6270	11827

	0	0
18	6639	11699
	0	0
19	7005	11564
	0	0
20	7366	11422
	0	0
21	7723	11273
	0	0
22	8076	11117
	0	0
23	8423	10955
	0	0
24	8765	10786
	0	0
25	9102	10611
	0	0
26	9434	10429
	0	0
27	9760	10241
	0	0
28	10080	10046
	0	0
29	10394	9846
	0	0
30	10702	9640
	0	0
31	11003	9427
	0	0
32	11297	9210
	0	0
33	11585	8986
	0	0
34	11866	8758
	0	0
35	12140	8524

	0	0
36	12406	8285
	0	0
37	12665	8040
	0	0
38	12916	7792
	0	0
39	13160	7538
	0	0
40	13395	7280
	0	0
41	13623	7017
	0	0
42	13842	6750
	0	0
43	14053	6479
	0	0
44	14256	6205
	0	0
45	14449	5926
	0	0
46	14635	5644
	0	0
47	14811	5359
	0	0
48	14978	5070
	0	0
49	15137	4778
	0	0
50	15286	4483
	0	0
51	15426	4186
	0	0
52	15557	3886
	0	0
53	15679	3584

```
            0          0
54      15791       3280
            0          0
55      15893       2973
            0          0
56      15986       2665
            0          0
57      16069       2356
            0          0
58      16143       2044
            0          0
59      16207       1732
            0          0
60      16261       1419
            0          0
61      16305       1104
            0          0
62      16340        789
            0          0
63      16364        474
            0          0
64      16379        158
            0          0
```

EXAMPLE SINE FUNCTION CALCULATIONS

input angle:

θ_{in} = \$03E8 = 00 000011 11101000

The quadrant is 00_2 → θ_{in} is a first quadrant angle

The index is 000011_2 → the index is \$3,therefore the fourth entry in TBL and DTBL will be used to calculate the angle

The fraction is 11101000_2 → this number will be used for interpolation

The form of TBL and DTBL:

index	TBL*	DTBL*
0	0	12867
(0°)	0	0
1	402	12859
(1.41°)	0	0
2	804	12843
(2.82°)	0	0
3	1205	12820
(4.23°)	0	0
4	1606	12789

(5.64°)	0	0

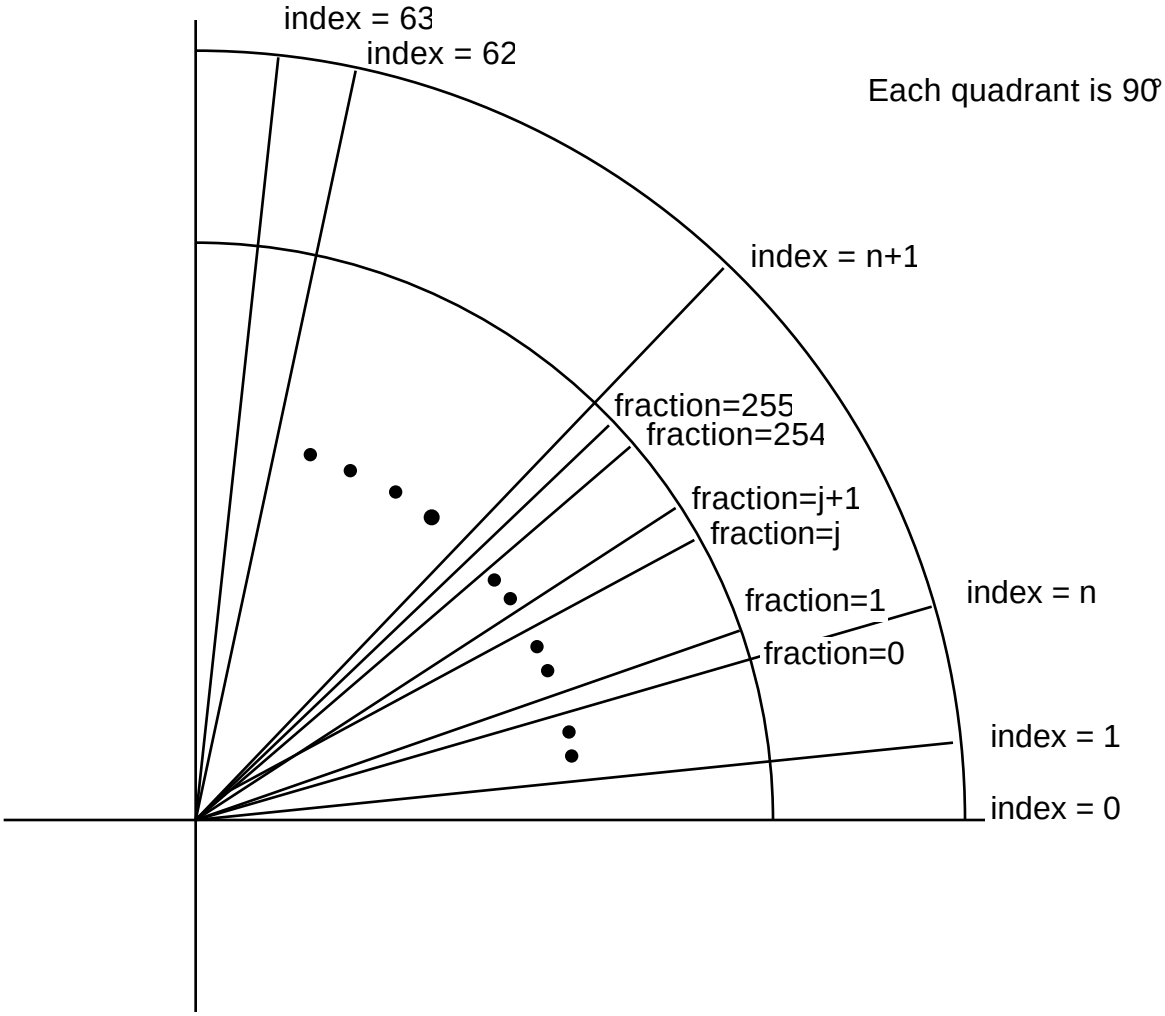
* TBL and DTBL are shown in decimal

CALCULATING THE SINE DIRECTLY FROM THE TABLE

From this table and using simple interpolation we can directly calculate the sine of $03E8.

To begin we determine the angle involved. Since index=3 the angle is between 4.23° and 5.64°, i.e. 4.23° plus some fraction of 1.41°. There are $FF increments between 4.23° and 5.64° so the fraction $E8 indicates that the fractional angle is $E8/$FF*1.41° or, after converting to decimal, 232/255*1.41° = 1.28°. The angle in question is then 4.23° (index=3) + 1.28° = 5.51°.

The answer is then sine(5.51°)*16384 = 1573.18 = $0625. Note that the sine must be multiplied by $2^{15}=16384_{10}$ to make sure that sine(90°) corresponds to bit 15 in the binary representation of the sine, i.e. $0100\ 0000\ 0000\ 0000_2$.



Distribution of angles using <quadrant, index,fraction> notation

CALCULATING THE SINE USING A TAYLOR SERIES

From this table the correct values of TBL and DTBL to be used for the calculation are:

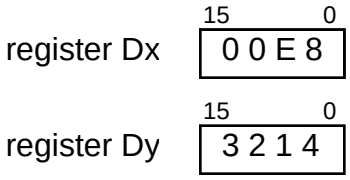
TBL($3) = 1205_{10} = $04B5

DTBL($3) = 12820_{10} = $3214

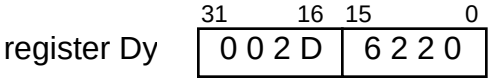
The first calculation is to multiply fraction times DTBL($3) using the MULU instruction

From θ_{in}, fraction is 11101000$_2$ = $E8. Assuming $E8 is in Dx and $3214 is in Dy, the result of the instruction MULU Dx,Dy will be

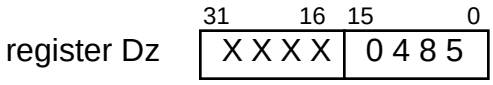
Before the MULU



After the MULU



To complete the calculation we need to add this to TBL($3). If we assume that we have put TBL($3) into register Dz we would have



The problem is that the original binary points of TBL and DTBL are NOT the same (see next page) so that we cannot directly add these numbers together. We will keep the format of TBL and shift the MULU result in register Dy so that the numbers can be correctly added together.

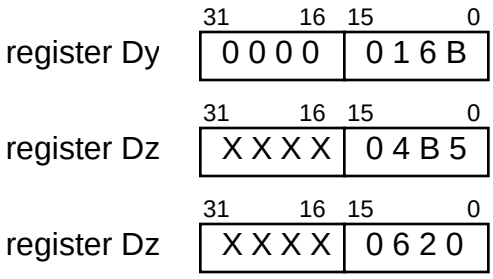
The number in Dy is

0000 0000 0010 1101 0110 0010 0010 0000_2

which needs to be shifted 13 bits to the right before it can be correctly added to Dz. A LSR of 13 bits results in the new contents of Dy:

0000 0000 0000 0000 0000 0001 0110 1011_2 = $016B

which can now be directly added to Dz (word length) to give $0620, the correct answer.



This was a first quadrant angle, sines in other quadrants can be computed using the appropriate trig identity:

In the second quadrant:

sine(x) = sine(180°-x) or, in hex, sine(x) = sine($8000-x)

In the third quadrant:

sine(x) = - sine(x-180°) or, in hex, sine(x) = - sine(x-$8000)

In the fourth quadrant:

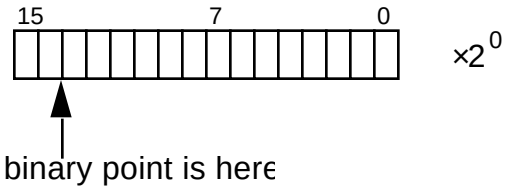
sine(x) = - sine(360°-x) or, in hex, sine(x) = - sine($10000-x)

NOTE: you should set the sign bit of your result after computing the angle.

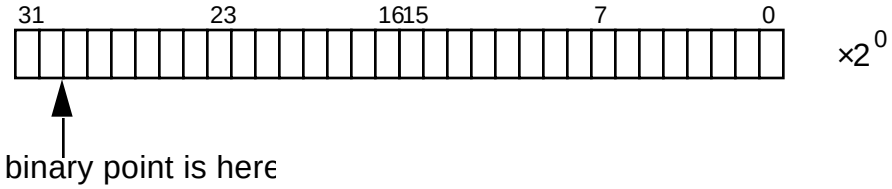
FIXED POINT ARITHMETIC

The math used in the sine lab is an example of fixed point arithmetic.

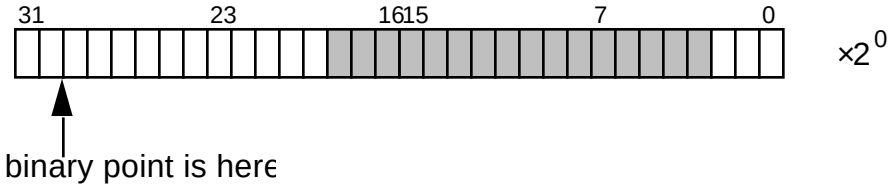
The numbers from TBL are in the format:



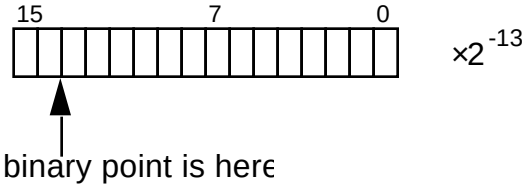
The original format for numbers from DTBL was:



This format was fine except that there were no 1's in any bit higher than bit 18. So, to save memory space I decided to only use the shaded bits, i.e.

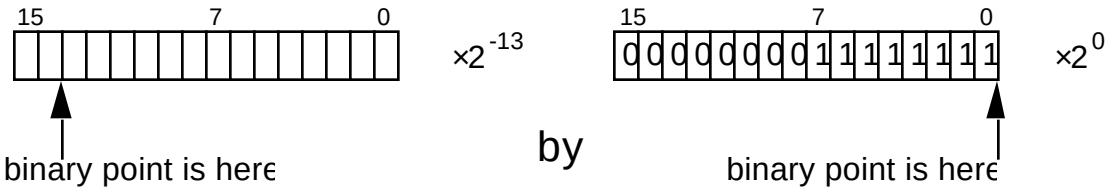


which are only 16 bits so they could be represented as a single word as shown below

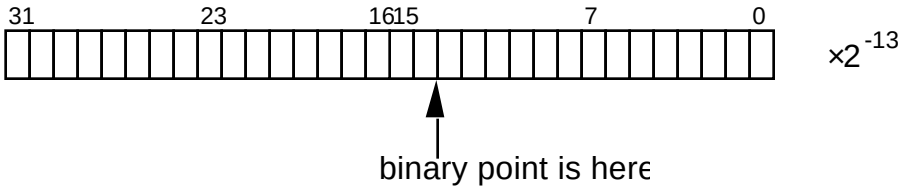


This format saves most of the significant information in DTBL.

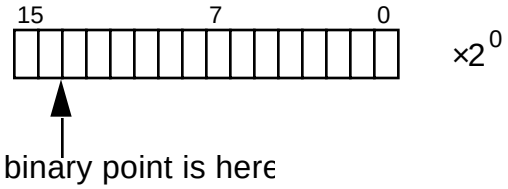
Now consider what happens when the offset is multiplied by this number from DTBL using a MULU instruction, i.e. multiply



to yield the 32-bit result

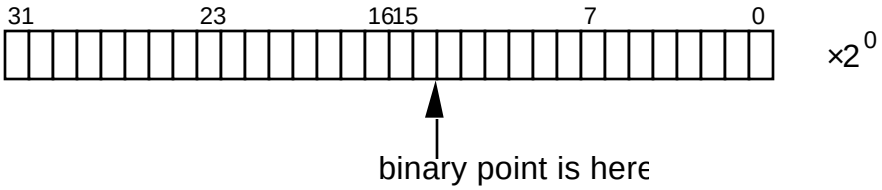


Now this cannot be directly added to the entry from TBL



because they are in different formats. The first thing to do is make the exponents the same, i.e. we have to get rid of the 2^{-13} exponent by shifting the number in the register 13 bits to the right. This requires a little thought—the location

of the binary point does not change as a result of this shift BUT the exponent has now become 2^0.



The format of the lower word in the register is the same as the entry from TBL so that the numbers can now be added together as 16-bit binary numbers in the format: