The Floating Point Package

For OSS Mac/65

by Louis J. Chorich III

OVERVIEW

The Floating Point Macro Library is designed to take advantage of the tremendous macro capabilities of the MAC/65 Macro Assembler produced by Optimized Systems Software. The Library is a programming tool intended to provide an interface between the programmer and the floating point routines built into the Atari ROM. Though assembly language provides only the capability to operate on integers, the use of the FP library allows a programmer to perform computations on numbers in the range of 10^-98 to 10^98. All computations are accurate to nine or ten digits depending upon the operation performed.

Included in the FP Library are the following capabilities:

Addition	2 Byte Integer to FP Conversion
Subtraction	FP to 2 Byte Integer Conversion
Multiplication	ASCII String to FP Conversion
Division	FP to ASCII String Conversion
Natural Logarithm	Input FP number from keyboard
Base 10 Logarithm	Print FP Number to Screen
Exponentiation	Comparison of Two FP Numbers
Inverse Logarithms	FP Array Management
Square Root	Move FP Numbers
Branching	Upon Comparison of two FP Numbers

The capabilities of the Trig library are contained in a separate section at the end of this documentation.

Included on the Library disk are eight files:

FP.LIB
FPANNOT.LIB
FPDEMO.M65
MSG.M65
FPDEMO.COM
TRIG.LIB
TRIGDEMO.M65
TRIGDEMO.COM

The FPANNOT.LIB file contains the fully commented source code while the FP.LIB file has the comments removed. FP.LIB functions identically to FPANNOT.LIB but occupies less disk space. FPDEMO.COM is the binary file produced by FPDEMO.M65 and MSG.M65. MSG.M65 is a part of the demo program. TRIG.LIB is the Trigonometric macros. TRIGDEMO.M65 and TRIGDEMO.COM are the source and object code of the program which demonstrates the use of the Trig macros. The Library disk is unprotected. You may make copies for your own use only.

This manual assumes that you are somewhat familiar with 6502 assembly language programming as well as the use of the MAC/65 Macro Assembler.

SYSTEM REQUIREMENTS

48K Atari Computer
Disk Drive
MAC/65 Macro Assembler

GETTING STARTED

In order to use the FP Library in your own programs, you first must .INCLUDE the FP.LIB or FPANNOT.LIB file in your source code. The best way to accomplish this is illustrated in the following example:

```
100  *=$4000          ;START OF CODE
110  JMP MYCODE
120  .INCLUDE #D:FP.LIB
130 ;YOU CAN INCLUDE ANY OTHER MACRO FILES YOU HAVE RIGHT HERE
140 MYCODE
150 ;YOUR PROGRAM STARTS HERE
```

Note the JMP MYCODE instruction in line 110. It is necessary that this is the first line after you indicate the address where assembly begins. If line 110 were not present, the first thing the program would do when run is execute code inside the FP library and certainly crash.

It might be mentioned that floating point numbers are stored in memory as six byte quantities. Every floating number requires six bytes of memory, no matter how small or large the value of the number.

THE FLOATING POINT MACRO LIBRARY

There are 25 macros in the FP library. They provide a complete interface to the Atari FP routines in ROM.

FPADD	LN	FPI	INPUTFP	FPCOMP
FPSUB	LOG	IFP	PRINTFP	BIGT
FPMUL	EXP	AFP	GETEL	BILT
FPDIV	EXP10	FASC	PUTEL	BIEQ
FPMOVE	EXPON	SQR	XXPUSH	XXPULL

FPADD

```
    Syntax: FPADD adr1, adr2, adr3
   Purpose: To add two FP numbers.
Parameters: adr1 - address of an FP number
            adr2 - another FP number
            adr3 - the FP result of the sum of the two numbers
   Example: FPADD REAL1,REAL2,RESULT
```

The FP number at REAL1 is added to the FP number at REAL2, and the result is stored at RESULT. REAL1 and REAL2 remain intact.

FPSUB

```
    Syntax: FPSUB adr1,adr2,adr3
   Purpose: To subtract two FP numbers.
Parameters: adr1 - address of an FP number
            adr2 - address of another FP number
            adr3 - the FP sum of the two numbers
   Example: FPSUB REAL1,REAL2,RESULT
```

The FP number at REAL2 is subtracted from the FP number at REAL1 and the result is stored at RESULT. REAL1 and REAL2 remain intact.

FPMUL

```
    Syntax: FPMUL adr1,adr2,adr3
   Purpose: To multiply two FP numbers.
Parameters: adr1 - address of a FP number
            adr2 - address of another FP number
            adr3 - address of the FP product
   Example: FPMUL AA,BB,XX
```

The FP number at AA will be multiplied by the FP number at BB and the FP product will be stored at XX. AA and BB remain intact.

FPDIV

```
    Syntax: FPDIV adr1,adr2,adr3
   Purpose: To divide one FP number by another.
Parameters: adr1 - address of the FP dividend
            adr2 - address of the FP divisor
            adr3 - address at which to store the FP result
   Example: FPDIV MILES,GALLONS,MILAGE
```

The FP number at MILES will be divided by the FP number at GALLONS and the result will be stored at MILAGE. MILES and GALLONS remain intact.

INPUTFP

```
    Syntax: INPUTFP adr
   Purpose: To get a FP number as input from the keyboard.
Parameters: adr - address at which to store the six byte
                  number entered at the keyboard
   Example: INPUTFP REAL
```

This macro waits for input from the keyboard, converts it to floating point notation, and stores it at address REAL. Memory starting at LBUFF ($580) is used as a buffer for keyboard input. Be careful! Any data in LBUFF may be overwritten!

PRINTFP

```
    Syntax: PRINTFP adr
   Purpose: To print a floating point number on the screen.
Parameters: adr - address of the FP number to print
   Example: PRINTFP REAL
```

PRINTFP is the complement of INPUTFP. PRINTFP takes the FP number at REAL, converts it to an ASCII string starting at LBUFF, and prints it to the screen. Like INPUTFP, PRINTFP uses LBUFF for workspace. Make sure that area of memory is free.

LN

```
    Syntax: LN adr1,adr2
   Purpose: To take the natural logarithm of a FP number.
Parameters: adr1 - address of the FP number whose natural
                   logarithm will be computed
            adr2 - address where the result will be stored
   Example: LN REAL1,REAL2
```

The natural log of the FP number at REAL1 will be computed and stored as a FP number at REAL2.

LOG

```
    Syntax: LOG adr1,adr2
   Purpose: To take the base 10 logarithm of an FP number.
Parameters: adr1 - address of the FP number whose base 10
                   logarithm will be computed
            adr2 - address where the result will be stored
   Example: LOG REAL1,REAL2
```

The base 10 log of the FP number at REAL1 will be computed and stored as a FP number at REAL2.

EXP

```
    Syntax: EXP adr1,adr2
   Purpose: To compute the inverse natural logarithm of a
            floating point number.
Parameters: adr1 - address of the FP number whose natural
                   log will be computed
            adr2 - address at which the result will be stored
   Example: EXP NUMBER,INVLOG
```

The inverse natural log of the FP number at address NUMBER is calculated and the result is stored starting at address INVLOG.

EXP10

```
    Syntax: EXP10 adr1,adr2
   Purpose: To calculate the inverse base 10 log of a
            floating point number.
Parameters: adr1 - address of the FP number whose inverse
                   base 10 log will be computed.
            adr2 - address at which to store the result
   Example: EXP10 TIME,VELOCITY
```

This macro calculates the inverse base 10 log of the FP number at TIME and stores the result at VELOCITY.

EXPON

```
    Syntax: EXPON adr1,adr2,adr3
   Purpose: To raise a FP number to a power.
Parameters: adr1 - address of the FP base
            adr2 - address of the FP exponent
            adr3 - address at which the FP result is stored.
   Example: EXPON BASE,EXPONENT,RESULT
```

This example calculates the value of BASE raised to the power EXPONENT and stores the floating point result in RESULT. This is the BASIC equivalent of RESULT = BASE^EXPONENT.

SQR

```
    Syntax: SQR adr1,adr2
   Purpose: TO calculate the square root of a FP number.
Parameters: adr1 - The address of a FP number
            adr2 - The address at which to store the result
                   of the square root calculation
   Example: SQR PRESSURE,ROOT
```

This macro determines the square root of the number at PRESSURE and stores the result at ROOT.

IFP

```
    Syntax: IFP int,adr
   Purpose: To convert a 2 byte integer to a 6 byte FP number.
Parameters: int - the address of the 2 byte integer
            adr - the address of the 6 byte FP result
   Example: IFP COUNT,REAL
```

The two byte integer at COUNT will be converted to a six byte FP number at REAL. The result can now be used in FP math calculations.

FPI

```
    Syntax: FPI adr,int
   Purpose: To convert a 6 byte FP number to a 2 byte integer.
Parameters: adr - address of the 6 byte FP number
            int - address of the 2 byte integer result
   Example: FPI REAL,COUNT
```

This macro is the complement of IFP. The FP number at REAL is converted into a two byte integer and stored at COUNT. The FP number is rounded up or down to the nearest whole number.

AFP

```
    Syntax: AFP string,adr
   Purpose: To convert an ASCII string of numeric characters.
            to FP notation.
Parameters: string - the address of the ASCII string
               adr - the address at which to store the FP result
   Example: AFP BUFFER,SCORE
```

Here, the ASCII string at BUFFER is converted to a FP number and stored at SCORE. The only legal characters in the ASCII string are numeric digits (0-9), a decimal point (.), a plus or minus sign (+/-), and an "E" when using scientific notation.
FASC

```
    Syntax: FASC adr1,adr2
   Purpose: To convert a floating point number to an
            ASCII string terminated by a carriage return.
Parameters: adr1 - The address of the FP number
            adr2 - The address at which to store the ASCII
                   string.
   Example: FASC NUM,BUFFER
```

The macro converts the floating point number stored at NUM to an ASCII string starting at address BUFFER and terminating with a carriage return.

FPMOVE

```
    Syntax: FPMOVE adr1,adr2
   Purpose: To move a six byte floating point number
            from one location to another.
Parameters: adr1 - the source address
            adr2 - the destination address
   Example: FPMOVE SUBTOT,TOTAL
```

This macro call will move the six byte FP number at SUBTOT into the six bytes at TOTAL. This effectively replaces TOTAL with SUBTOT. At the end of the macro call, the same FP number will be at SUBTOT and TOTAL.

FPCOMP

```
     Syntax: FPCOMP adr1,adr2,adr3
    Purpose: To compare two floating point numbers and determine
             if they are equal or if the first is less than the
             second or if the first is greater than the second.
Parameters: adr1 - the address of the first FP number
            adr2 - the address of the second FP number
            adr3 - the address of the byte at which to store
                   the result of the comparison
   Example: FPCOMP REAL1,REAL2,FLAG
```

The floating point number at REAL1 is compared to the floating point number at REAL2 and the one byte result of the comparison is stored in FLAG as follows:

```
     1 = REAL1 > REAL2
     0 = REAL1 = REAL2
   255 = REAL1 < REAL2
```

Notice that if REAL1 is greater than REAL2, the result stored in FLAG is positive. If REAL1 is equal to REAL2, the result stored in FLAG is zero. If REAL1 is less than REAL2, the result stored in FLAG is negative (to the 6502, all numbers from 128 to 255 [$7F - $FF] are considered negative. This makes for easy testing of the result byte and branching accordingly.

BIEQ

```
    Syntax: BIEQ adr1,adr2,adr
   Purpose: To test if two FP numbers are equal
            and branch to a given address if the condition
            is true.
Parameters: adr1 - The first FP number
            adr2 - The second FP number
            adr3 - The address at which to branch if the
                   two FP numbers are equal
  Example: BIEQ COUNT,MAX,EXIT
```

BIEQ stands for "Branch If Equal." In this case, if the FP number at COUNT is equal to the FP number at MAX, then the program will branch to EXIT. The assembly language equivalent of this is "JMP EXIT." Of course you do not have to use a label. Specifying an absolute address such as $0600 in the macro call would be perfectly acceptable.

BIGT

```
    Syntax: BIGT adr1,adr2,adr3
   Purpose: To test if one FP number is greater than
            another and branch to a given address if the
            condition is true.
Parameters: adr1 - The first FP number
            adr2 - The second FP number
            adr3 - The address at which to branch
   Example: BIGT REAL1,REAL2,GETNEXT
```

BIGT stands for "Branch If Greater Than." In this case, if the FP number at REAL1 is greater than the FP number at REAL2, the program will branch to the label GETNEXT   (JMP GETNEXT).

BILT

```
    Syntax: adr1,adr2,adr3
   Purpose: To test if one floating point number is less than
            another and to branch to a given address if the
            condition is true.
Parameters: adr1 - The address of the first FP number
            adr2 - The address of the second FP number
            adr3 - The address at which to branch
   Example: BILT SCORE,MIN,$0680
```

If the FP number at SCORE is less than the FP number at MIN, the program will branch to address $0680 (JMP $0680). Otherwise, the program will continue unaffected.

PUTEL

```
    Syntax: PUTEL adr1,adr2,adr3
   Purpose: To put a floating point number into any element
            of an array of floating point numbers.
Parameters: adr1 - Base address of the array of FP numbers
            adr2 - This parameter is the memory address
                   whose value holds the number (0-255)
                   corresponding to the element in the FP
                   array into which the FP number at adr3
                   is to be placed
            adr3 - The address of the FP number which is to
                   inserted into the FP array
   Example: PUTEL ARRAY,ELEMENT,REAL
```

This macro takes the FP number at REAL and inserts it into the ELEMENTth element of the FP array starting at address ARRAY. Note that the first ELEMENT is ELEMENT 0, the second is ELEMENT 1, etc. A floating point array does not have to be specially declared or dimensioned. All you need to do is set aside enough room to hold your array. Each element of a floating point array consumes six bytes. If you only set aside sixty bytes for a floating point array, you must be careful not to access any element outside the range of 0 through 9. The zeroth element of the array is located at ARRAY. Element 1 is located at ARRAY+6, and element 2 is located at ARRAY+12, etc.

Take note that this macro reads the value in address ELEMENT to determine into which element of the FP array REAL will be placed. Suppose you wanted to put the FP number at REAL into the tenth element of the floating point array starting at ARRAY. First, you must set aside 66 bytes to hold the entire 11 element (0-10) array. This is accomplished in the following manner:

```
     ARRAY *=*+66
```

This line of assembly code defines the base address of the array as ARRAY and sets aside 66 bytes of memory to hold the array.

To put REAL in the tenth element of the array, there is a correct and incorrect method:

```
     Incorrect: PUTEL ARRAY,10,REAL

       Correct: LDA #10
                STA ELEMENT
                PUTEL ARRAY,ELEMENT,REAL
```

The first macro call would yield unpredicatable results, but the second will always place the FP number at REAL into the tenth element of the array. The BASIC equivalent of the above macro is:

```
     ARRAY(ELEMENT)=REAL
```

Refer to the demo programs for detailed examples of implementing floating point arrays.

GETEL

```
    Syntax: GETEL adr1,adr2,adr3
   Purpose: To retrieve any element from an array of floating
            point numbers.
Parameters: adr1 - The base address of the floating point array
            adr2 - The address of the memory location which
                   holds the number (0-255) which corresponds
                   to the number of the element of the array
                   which is to be extracted and placed in
                   adr3
            adr3 - The address at which to place the FP
                   number extracted from the array
   Example: GETEL ARRAY,ELEMENT,REAL
```

GETEL is the complement of PUTEL. This macro will take the ELEMENTth element of the FP array starting at address ARRAY and place it at REAL. The same restrictions that apply to PUTEL apply to GETEL. The BASIC equivalent of the above macro is:

```
     REAL=ARRAY(ELEMENT)
```

You may notice the ease with which this macro can be placed in a loop. Each pass through the loop, the value in element may be incremented to, for example, print an entire array of floating point numbers.

XXPUSH

```
    Syntax: XXPUSH
   Purpose: To save the X, Y, and Accumulator registers on
            the stack before calling a macro
Parameters: None
   Example: XXPUSH
```

This macro is used internally by the Floating Point Library and need not be used by you when implementing any floating point macros. XXPUSH and its complement, XXPULL, make sure that your X, Y, and Accumulator registers are not altered by any the floating point macros. If you for any reason use this macro, make sure that you use XXPULL to pull the X,Y, and A registers off the stack at the appropriate time, or else havoc will surely prevail. Use at your own risk!!!

XXPULL

```
    Syntax: XXPULL
   Purpose: To restore the previously saved X, Y, and
            Accumulator registers.
Parameters: None
   Example: XXPULL
```

XXPULL is the complement of XXPUSH as explained above. You can see how XXPUSH and XXPULL are used as the first and last lines of every macro definition in the Floating Point Macro Library.

LIMITATIONS OF THE FP LIBRARY

Raising a negative number to a power by calling the EXPON macro is likely to yield unpredicatable results. This is due to the fact that the calculation involves the use of logarithms, and negatives numbers do not have logarithms associated with them. The Floating Point Library makes no attempt to catch such error. Your program will not crash if you try to do something impossible, such as take the square root of a negative number, but the results will be unpredicatable.

PROGRAMMING TECHNIQUES

A major criticism of the use of macros in assembly language is the fact that large macros tend to consume memory ravenously if used heavily. The reason for this is that a macro is not like a subroutine or some type of function which pulls its parameters off a software stack. A macro is expanded, or reproduced, in its entirety each time it is called.

A simple solution to this problem is to implement heavily used macros as assembly language subroutines. The only disadvantage of this technique is that the parameters passed to the macro are fixed. For example, the following two lines of code are all that is necessary to create a subroutine that prints a floating point number.

```
     PRINT PRINTFP REAL
           RTS
```

The routine can be called by the following instruction and consumes only three bytes each time it is called.

```
     JSR PRINT
```

Of course, you are limited in that the only floating point number this routine can print is the one at address REAL. But if you specify real as target address at which to store the result of any FP calculation you intend to print, then everything is set up automatically and a JSR to the PRINT subroutine is all that is necessary to print the result of your calculations. Although not appropriate under all circumstances, this technique can save a considerable amount of memory when used efficiently.

The Demo Program

Included on the diskette is a the source and object code for a demo of the Floating Point Library, FPDEMO.COM. The source code for the demo program is in two files, FPDEMO.M65 and MSG.M65. The RUN address for FPDEMO.COM is $5000

The Trigonometric Library

by Louis J. Chorich III

OVERVIEW

The Trig Library accompanies the Floating Point Macro Library. It contains an additional eight macros which implement the basic trigonometric functions sine, cosine, tangent, cotangent, secant, and cosecant. Also included are macros for degree/radian conversions.

The Trig Library is a stand alone library of macros and may be used independently of the Floating Point Library or in conjunction with it.

All trig macros operate upon angles measured in radians. Specifying an angle in degrees will return incorrect results. All values must be in 6 byte Floating Point format.

The Trig Library is called TRIG.LIB and is included on the Library diskette. To access the trig macros, it is necessary to INCLUDE the library file in your source code in the same way that FP.LIB is INCLUDEed. If both libraries are needed, it is a good idea to INCLUDE them together at the beginning of your assembly program. This is explained fully in the "Getting Started" section the Floating Point Package manual.

Included on the diskette is a demo of the Trig Library, TRIGDEMO.COM. The source code is TRIGDEMO.M65. The RUN address for TRIGDEMO.COM is $5000.

THE TRIGOMOMETRIC MACROS

SIN

```
    Syntax: SIN adr1,adr2
   Purpose: To calculate the sine of an angle.
Parameters: adr1 - address of a FP angle in radians
            adr2 - address at which to store the FP result
   Example: SIN ANGLE,RESULT
```

This macro will calculate the sine of the angle represented by the 6 byte floating point number at ANGLE and store the result of its calculation at RESULT.

COS

```
    Syntax: COS adr1,adr2
   Purpose: To calculate the cosine of an angle.
Parameters: adr1 - address of a FP angle in radians
            adr2 - address at which to store the FP result
   Example: COS ANGLE,RESULT
```

The cosine of the floating point number at ANGLE is evaluated and the result is stored at RESULT.

TAN

```
    Syntax: TAN adr1,adr2
   Purpose: To calculate the tangent of an angle.
Parameters: adr1 - address of a FP angle in radians
            adr2 - address at which to store the FP result
   Example: TAN RADIANS,TANGENT
```

This macro will compute the tangent of the angle at RADIANS and store the floating point result at TANGENT.

COT

```
    Syntax: COT adr1,adr2
   Purpose: To calculate the cotangent of an angle.
Parameters: adr1 - address of a FP angle in radians
            adr2 - address at which to store the FP result
   Example:  COT RADIANS,COTAN
```

The cotangent of the angle at RADIANS is evaluated and the floating point result is stored at COTAN.

SCN

```
    Syntax: SCN adr1,adr2
   Purpose: To calculate the secant of an angle.
Parameters: adr1 - address of a FP angle in radians
            adr2 - address at which to store the FP result
   Example: SCN THETA,SECANT
```

This macro call will calculate the secant of the radian angle at THETA and store the result at SECANT.

CSC

```
    Syntax: CSC adr1,adr2
   Purpose: To calculate the cosecant of an angle.
Parameters: adr1 - address of a FP angle in radians
            adr2 - address at which to store the result
   Example: CSC ALPHA,RESULT
```

This macro will calculate the cosecant of the radian angle at ALPHA and store the cosecant of ALPHA at RESULT.

TORAD

```
    Syntax: TORAD adr1,adr2
   Purpose: To convert a degree angle to radians.
Parameters: adr1 - the address of a FP angle in degrees
            adr2 - the address at which to store the FP angle
                   in degrees
   Example: TORAD DEGREES,RADIANS
```

This macro is used to convert the degree angle at DEGREES to a radian angle at RADIANS.

TODEG

```
    Syntax: TODEG adr1,adr2
   Purpose: To convert a radian angle to degrees.
Parameters: adr1 - the address of a FP angle in radians
            adr2 - the address at which to store the FP angle
                   in radians
   Example: TODEG RADIANS,DEGREES
```

This macro is used to convert the radian angle at RADIANS to a degree angle at DEGREES.