EXTENDED MATH PACKAGE

for SUPER-FORTH 64™

By Bruce Jordan

Available at the Vintage Volts website

http://www.vintagevolts.com

INTRODUCTION

The SUPER-FORTH Math Extension Package is a group of mathematical utility packages designed to enhance SUPER-FORTH. These utilities are provided as source screen to be loaded into the SUPER-FORTH system.

THE PACKAGE CONTAINS:

1. Extensive floating point math package.

2. 2-dimensional matrix word set.

3. Multi-dimensional matrix word set.

4. The ALGEBRAIC EXPRESSION EVALUATOR.

5. An extended floating point algebraic word set.

NOTE : The symbol <CR> appears throughout this manual. This symbol indicates that you are to type a carriage return.

FLOATING POINT MATH PACKAGE

for SUPER-FORTH 64™

By Bruce Jordan

SUPER FORTH FLOATING POINT PACKAGE

The SUPER-FORTH Floating Point Package is a quality floating point system that contains the standard floating point word set plus many higher level mathematical functions and FORTH oriented extensions that make it a valuable programming companion for SUPER-FORTH. The SUPER-FORTH Floating Point Package combines the features of fast execution time, flexibility, minimum memory requirements and high accuracy.

LIST OF FEATURES

RAPID EXECUTION.

LOW MEMORY REQUIREMENTS.

EASY NUMBER CONVERSION: single to floating point numbers
floating point to single numbers
double to floating point numbers
floating point to double numbers
strings to floating point numbers
floating point numbers to strings

NUMBERS CAN BE ENTERED IN STANDARD DECIMAL FORM OR SCIENTIFIC NOTATION FORM.

WIDE NUMBER RANGE.

approximately +-2E+38 to +-3E-39

ACCURACY TO EIGHT DECIMAL PLACES.

HIGHER LEVEL MATH FUNCTIONS.

FLOATING POINT STACK MANIPULATION WORDS.

FLOATING POINT BOOLEAN OPERATORS.

NUMBER CONVERSION

F->S	FLOATING POINT TO SINGLE NUMBER
S->F	SINGLE TO FLOATING POINT NUMBER
F->D	FLOATING POINT TO DOUBLE NUMBER
D->F	DOUBLE TO FLOATING POINT NUMBER
$->F	STRING TO FLOATING POINT NUMBER
F->$	FLOATING POINT NUMBER TO STRING

FDROP	FLOATING POINT DROP
FDUP	FLOATING POINT DUP
F2DUP	FLOATING POINT 2DUP
FOVER	FLOATING POINT OVER
FPICK	FLOATING POINT PICK
FROLL	FLOATING POINT ROLL
FROT	FLOATING POINT ROT
FSWAP	FLOATING POINT SWAP
F.	PRINT A FLOATING POINT NUMBER
E!	ENTER EXPONENT

MATH FUNCTIONS

F+	ADD
F-	SUBTRACT
F*	MULTIPLY
F/	DIVIDE
F^	POWER FUNCTION
1/F	RECIPROCAL
FABS	ABS
FMAX	MAXIMUM
FMIN	MINIMUM
FNEGATE	NEGATE
FRAC	DECIMAL FRACTION
FSQR	SQUARE ROOT
FCOS	COSINE
FSIN	SINE
FTAN	TANGENT
FACOS	ARC COSINE
FASIN	ARC SINE
FATN	ARC TANGENT
FE^X	'e' TO THE X POWER
FSGN	SIGNUM FUNCTION
FLOG	NATURAL LOG
FLOGX	LOG TO THE BASE X
PI	CONSTANT FOR PI
EX	CONSTANT FOR 'e'

FLOATING POINT MEMORY WORDS

FVARIABLE	CREATE FLOATING POINT VARIABLE
FCONSTANT	CREATE FLOATING POINT CONSTANT
F@	FETCH FLOATING POINT VARIABLE
F!	STORE FLOATING POINT VALUE

LOGICAL AND COMPARISON WORDS

FOR	FLOATING POINT OR
FAND	FLOATING POINT AND
F<	FLOATING POINT <
F=	FLOATING POINT =
F>	FLOATING POINT >

STACK MANIPULATORS

?FDEPTH	FLOATING POINT ?DEPTH
FDEPTH	FLOATING POINT DEPTH

To load the SUPER-FORTH Floating Point Package...

1. Load and run SUPER-FORTH.
2. Insert the SUPER-FORTH EXTENDED MATH-I/O UTILITY DISK into your disk drive.
3. Type: 2 LOAD

This will load the entire Floating Point Package.

USING THE SUPER-FORTH FLOATING POINT PACKAGE

Every attempt has been made to make the SUPER-FORTH Floating Point Package as easy to use and trouble free as possible. However, there's one thing that you should always remember: Always type FLP-INIT or FLP-EXIT immediately after any reset that occurred while in Floating Point Mode. A reset from Floating Point Mode leaves you someplace between FLP-INIT and FLP-EXIT; and things can get weird out there. FLP-INIT or FLP-EXIT will restore everything properly.

Now, let's try out the SUPER-FORTH Floating Point Package.

First, type:

```
FLP-INIT
```

You are now in Floating Point Mode. This allows you to enter floating point numbers, and use them in calculations and colon definitions.

Now, type:

```
123.  45.  F^  F.
```

You just got an error message, right? I had you do this to show you that there is limits to the size of the numbers you can use. The fortyth power of one hundred and twenty three is a big number! Far too big for the Floating Point Package to handle. The general rule is a result of a calculation can be no greater than +-1.70141183E+38, or less than +-2.93873588E-39.

Before you go any farther, type FLP-INIT once again. You are now back in Floating Point Mode.

Now, let's try a few simple floating point operations. First, let's enter a few floating point numbers.

EXAMPLE :

```
1.2  3.4  5.  <CR> OK
```

There are now 3 floating point numbers on the stack. Let´s print the top floating point number:

```
F.  <CR> 5 OK
```

Next, we´ll add the remaining two floating point numbers together:

```
F+  <CR> OK      ( 1.2 AND 3.4 ARE ADDED ON THE STACK. )
```

If we print the result, we get:

```
F.  <CR>  4.6 OK
```

You can see that math operations with The Extended Floating Point Math Package are carried out much the same as when doing math operations in fixed-point form. The only difference is that each operation (+ - * / ...) is preceded by the letter F (F+ F- F* F/ ...).

Here´s another example. We will subtract one number from another, and then multiply the result by a third number.

EXAMPLE :

```
FLP-INIT

4.5  2.3  F-  10.11  F*  F.  <CR>  22.24 OK
```

Next, let´s try using some of the floating point routines within a colon definition.

The main things to remember when using floating point numbers within colon definitions are:

1. Make sure you´re in Floating Point Mode.

2. FLP-INIT and FLP-EXIT should not be used within a colon definition.

Other than these restrictions, using floating numbers is alot like using double numbers in SUPER-FORTH.

Let´s try a definition:

We'll define a word that takes the square root of the numbers 1-10.

```
FLP-INIT

: 10SQROOT
CR  11  1  DO  I  S->F  ( CONVERT INDEX TO FLOATING # )
FSQR  F.                ( GET SQUARE ROOT AND PRINT IT )
CR  LOOP ;
```

Now, when we type :

```
10SQROOT  <CR>
1
1.41421356
1.73205081
2
2.23606798
2.44948974
2.64575131
2.82842713
3
3.16227736
```

Notice that we first had to convert the single number index of the loop to a floating point number before using it with floating point math routine routine FSQR.

Now let's define a word that displays the flexibility of the Extended Floating Point Math Package. We'll define a word that illustrates entering floating point numbers from the keyboard,

```
FLP-INIT ( YOU DON'T HAVE TO TYPE THIS EVERY TIME YOU )
         ( DEFINE  A WORD, JUST SO LONG AS YOU KNOW )
         ( YOU'RE IN  FLOATING POINT MODE. )

:  LOGGER
CR  ." GIVE ME A NUMBER >0 "
$INPUT          ( GET A STRING FROM THE KEYBOARD )
CR
$->F            ( CONVERT IT TO A FLOATING POINT NUMBER )
FDUP
FLOG  FSWAP  CR
." THE NATURAL LOG OF "  F.
." IS "  F.  ;
```

Now try...

```
LOGGER  <CR>  GIVE ME A NUMBER  >0   21. <CR>

THE NATURAL LOG OF 21 IS  3.04452244 OK
```

It´s possible to use floating point numbers in the setting of control parameters of control words such as DO loops:

```
:  LOOPIT ( LOOP N NUMBER OF TIMES )
F->S      ( CONVERT FLOATING NUMBER TO SINGLE NUMBER )
0  DO  I  .  LOOP  ;
```

Give LOOPIT a floating point number for the limit, and...

```
23.456  LOOPIT   <CR>   0 1 2 3 4 5 6 7 8 9 10 11 12 13
14 15 16 17 18 19 20 21 22 OK
```

The next example shows how to us the Floating Point Comparison words. The example tests to see if one floating point number is the same as another floating point number.

```
FLP-INIT ( JUST CHECKING )

:  TESTER  ( TEST A NUMBER )
FDUP
123.00021  F<  IF  ." GO BIGGER "  FDROP
ELSE  123.00021  F=  IF  ." YOU GOT IT!!!"
ELSE  ." GO SMALLER "
THEN
THEN ;
```

Let´s try our test.

```
123.  TESTER  <CR>  GO BIGGER OK
1000.  TESTER  <CR>  GO SMALLER OK
123.00020  TESTER  <CR>  GO BIGGER OK
123.00021  TESTER  <CR>  YOU GOT IT!!! OK
```

Floating point numbers can be entered into colon definitions without being in floating point mode. The easiest example for this is in the definition of a floating point constant. What we want is a word that when called will leave some value on the stack. We can do this in several different ways.

First, we could define a constant using FCONSTANT:

```
FLP-INIT ( YEAH, YEAH. I KNOW...)

6.626  -34  E!  FCONSTANT  PLANK´S  <CR>
```

Now, whenever we type PLANK´S, we get...

```
PLANK´S  <CR>

F.  <CR>  6.62599999E-34 OK
```

NOTE : There´s a slight inaccuracy in the Floating Point Package. However, this error is so small that it should not be a problem for most applications.

The second way to create a constant is to define a word who´s definition is just the value to be left on the stack:

```
: TEMP1  3.0  ;
```

Now, when we type TEMP1, we get...

```
TEMP1  F. <CR>  3 OK
```

But, let´s say you´re not in Floating Point Mode, and you want to define a word that will return a floating point number. How do you do it? Easy, just enter three single numbers that are equivalent to the floating point value (See <FNUM>).

EXAMPLE :

First we find out what a floating point three looks like when displayed as three single numbers.

```
FLP-INIT

3.  .S   <CR>  0  192  130 OK

FLP-EXIT ( LEAVE FLOATING POINT MODE. )
```

So our constant word will use the above three numbers:

```
: TEMP2  0  192  130  ;
```

When we use TEMP2 in floating point mode, we get:

```
FLP-INIT

TEMP2  F.  <CR>  3 OK
```

This last example was merely intended to give a better understanding of how the SUPER-FORTH Floating Point Package handles floating point numbers.

ERROR MESSAGES

Some floating point operations will cause the SUPER-FORTH system to reset if improper values are used. For instance, trying to obtain the natural log of zero will cause a system reset. When this happens, an error message will be displayed to give you an idea of what it was you did wrong.

?ILLEGAL QUANTITY :

An illegal operation was attempted such as taking the root of a negative number, or evaluating the natural log of zero.

?OVERFLOW ERROR :

An operation was attempted who´s result is larger than the largest number allowed: 1.70141884E+38.

As a final reminder:

1. Always enter floating point numbers with a decimal point, even zero (0.).
2. Almost all floating point commands start with F. Watch out for using DUP when what you want is FDUP, or DROP when you really meant FDROP.
3. Always be sure of what mode you´re in.
4. Always re-enter or exit Floating Point Mode on a reset.
5. Colon definitions using floating point numbers must be defined within Floating Point Mode.
6. Division by zero (0) will give erroneous results.

FLOATING POINT WORD SET

The following is a description of the SUPER-FORTH Floating Point Package word set.

NOTE : It is not necessary to type FLP-INIT each time you perform a floating point operation. At the risk of being redundant, many of the examples contain the command FLP-INIT. This was intended only as a reminder that you must be in floating point mode to do floating point operations.

FLP-INIT : INITIALIZE FLOATING POINT MODE

(---)

This word places you in floating point mode.

This word replaces the number conversion routine <NUMBER> with <FNUM>, replaces DLITERAL with FLITERAL, assigns INBUF as the System Input Buffer and changes the system WARM-START vector.

FLP-EXIT : EXIT FLOATING POINT MODE

(---)

This word is used to leave floating point mode.

This word restores <NUMBER> as the number conversion routine, replaces DLITERAL, restores $100 as the Input Buffer area and resets the WARM-START vector.

F- : FLOATING POINT SUBTRACTION

(f1 f2 --- f3)

Subtract f2 from f1 leaving the difference f3 on the stack.

EXAMPLE :

```
-23.4  5.6  F-  F.  <CR>  -29 OK

14.  2.5  F-  F.  <CR>  11.5 OK
```

F+ : FLOATING POINT ADDITION

(f1 f2 --- f3)

Add f2 and leaving the sum f3 on the stack.

EXAMPLE :

```
3.2  1.5  F+  F.  <CR>  4.7 OK
```

F/ : FLOATING POINT DIVISION

(f1 f2 --- f3)

Divide f2 by f1 leaving the quotient f3 on the stack.

EXAMPLE :

```
124.5  7.2  F/  F.  <CR>  17.2916667 OK
```

NOTE : Division by zero will give erroneous results.

F* : FLOATING POINT MULTIPLICATION

(f1 f2 --- f3)

Multiply f2 by f1 leaving the product f3 on the stack.

EXAMPLE :

```
-1.234  3.2  F*  F.  <CR>  -3.9488 OK
```

F. : PRINT FLOATING NUMBER

(f ---)

Prints the floating point number on the top of the stack.

EXAMPLE :

```
FLP-INIT
23.345  F.  <CR>  23.345 OK
```

E! : ENTER EXPONENT

(f n --- f)

Allows the entering of an exponent for a floating point number.

EXAMPLE 1 :

```
1.665  -15  E!  F.  <CR>  1.665E-15 OK
```

EXAMPLE 2 :

```
23.221  24  E!  <CR>
F.  <CR>  23.221E+24 OK
```

FABS : FLOATING POINT ABSOLUTE VALUE

(f --- f)

Returns the absolute value of the floating point number on the stack.

EXAMPLE :

```
FLP-INIT
-1.234567  FABS  F.  <CR>  1.234567 OK
```

FNEGATE : FLOATING POINT NEGATE

(f --- -f)

Leaves the negative of the floating point number on the stack.

EXAMPLE :

```
-123.456  FNEGATE  F.  <CR>  123.456 OK
123.456  FNEGATE  F.  <CR>  -123.456 OK
```

FMAX : FLOATING POINT MAXIMUM

(f1 f2 --- f3)

Leaves the greater of two floating point numbers on the stack.

EXAMPLE :

```
12.003  12.001  FMAX  F.  <CR>  12.003 OK
```

FMIN : FLOATING POINT MINIMUM

(f1 f2 --- f3)

Leaves the lesser of two floating point numbers on the stack.

EXAMPLE :

```
.0321  1.23  FMIN  F.  <CR>  .0321 OK
```

FINT : RETURN INTEGER

(f --- f)

Returns the integer portion of a floating point number.

EXAMPLE :

```
12.3456  FINT  F.  <CR>  12 OK
```

FRAC : RETURN FRACTION

(f --- f)

Returns the fraction portion of a floating point number.

EXAMPLE :

```
123.456   FRAC  F.  <CR>  .45599997
```

FCOS : RETURNS COSINE

(f --- f)

Returns the cosine of the floating point number, where the floating point number represents an angle in radians.

EXAMPLE :

```
1.34  FCOS  F.  <CR>  .228752808 OK
```

FSIN : RETURNS SINE

(f --- f)

Returns the sine of the floating point number, where the floating point number represents an angle in radians.

EXAMPLE :

```
1.34  FSIN  F.  <CR>  .973484542 OK
```

FTAN : RETURNS TANGENT

(f --- f)

Returns the tangent of the floating point number, where the floating point number represents an angle in radians.

EXAMPLE :

```
1.34  FTAN  F.  <CR>  4.25561789 OK
```

FACOS : RETURNS ARC COSINE

(f --- f)

Returns the arc cosine of the floating point. The result is in radians.

EXAMPLE :

```
.221  FACOS  F.  <CR>  1.34795662 OK
```

@(FASIN : RETURNS ARC SINE)

(f --- f)

Returns the arc sine of the floating point. The result is in radians.

EXAMPLE :

```
.221  FASIN  F.  <CR>  .222839705 OK
```

FATN : RETURNS ARC TANGENT

(f --- f)

Returns the arc tangent of the floating point number. The result is in radians.

EXAMPLE :

```
.221  FATN  F.  <CR>  .217503939 OK
```

FSQR : FLOATING POINT SQUARE ROOT

(f --- f)

Returns the square root of a floating point number.

EXAMPLE :

```
64.  FSQR  F.  <CR>  8 OK
```

F^ : POWER FUNCTION

(f1 f2 --- f3)

Raises f1 to the f2th power leaving the result f3 on the stack.

EXAMPLE :

```
2.  5.  F^  F.  <CR>  32 OK
```

FE^X : e TO THE POWER OF X

(f --- f)

Returns e (2.71828183) raised to fth power, where f is a floating point number.

EXAMPLE :

```
21.  FE^X  F.  <CR>  1.31881573E+9 OK
```

FLOG : NATURAL LOG FUNCTION

(f --- f)

Returns the natural log of a floating point number.

EXAMPLE :

```
1.31881573  9  E!  FLOG  F.  <CR>  21 OK
```

FLOGX : LOG TO BASE X

(f1 f2 --- f3)

Returns the log of f1 to the f2 base, leaving the result f3 on the stack.

EXAMPLE :

```
5.  2.  FLOGX  F.  <CR>  2.32192809 OK
```

FSGN : SIGNUM FUNCTION

(f --- f)

Returns -1 if f is less than 0, +1 if f if greater than 0 and 0 if f equals 0.

EXAMPLE :

```
-123.456  FSGN  F.  <CR>  -1 OK
```

FAND : FLOATING POINT LOGICAL AND

(f1 f2 --- f3)

ANDs two floating point numbers in the range of +-32767, returning the result on the top of the stack.

EXAMPLE :

```
123.456  37.5  FAND  F.  <CR>  33 OK
```

FOR : FLOATING POINT LOGICAL OR

(f1 f2 --- f3)

ORs two floating point numbers in the range of +-32767, returning the result on the top of the stack.

EXAMPLE :

```
123.456  37.5  FOR  F.  <CR>  127 OK
```

1/F : RECIPROCAL

(f --- f)

Returns the reciprocal of a floating point number.

EXAMPLE :

```
100.  1/F  F.  <CR>  .01 OK
```

F< : FLOATING POINT <

(f1 f2 --- flag)

Returns a single number 1 if f1 is less than f2, and a 0 if f1 is equal to or greater than f2.

F= : FLOATING POINT =

(f1 f2 --- flag)

Returns a single number 1 if f1 is equal to f2, and a 0 if f1 is less than or greater than f2.

F> : FLOATING POINT >

(f1 f2 --- flag)

Returns a single number 1 if f1 is greater than f2, and a 0 if f1 is less than or equal to f2.

PI : CONSTANT

(---)

Returns the value of pi on the stack.

EXAMPLE :

```
PI  F.  <CR>  3.14159265 OK
```

EX : CONSTANT

(---)

Returns the value of e on the stack.

EXAMPLE :

```
EX  F.  <CR>  2.71828138 OK
```

FCONSTANT : FLOATING POINT CONSTANT

(f ---)

Creates a floating point constant, who's value is placed on the top of the stack when its name is called.

EXAMPLE :

```
1.234  FCONSTANT  ABC  <CR>
ABC  F.  <CR>  1.234 OK
```

FVARIABLE : FLOATING POINT VARIABLE

(---)

Creates a floating point variable.

EXAMPLE :

```
FVARIABLE  VAR1  <CR>
```

Created a floating point variable named VAR1.

F! : FLOATING POINT STORE

(f addr ---)

Stores a floating point number in memory.

EXAMPLE :

```
FVARIABLE  VAR1  <CR>   ( CREATE A VARIABLE. )
123.456  VAR1  F!  <CR>
```

Stores 123.456 in floating point variable VAR1.

F@ : FLOATING POINT FETCH

(addr --- f)

Fetches a floating point value from memory.

EXAMPLE :

```
FVARIABLE  VAR1  <CR>
123.456  VAR1  F!  <CR>
VAR1  F@  F.  <CR>  123.456 OK
```

FDROP : FLOATING POINT DROP

(f1 f2 --- f1)

Drops the top floating point number from the stack.

EXAMPLE :

```
123.45  23.45  FDROP  <CR>
```

Leaves 123.45 on the stack and drops 23.45.

FDUP : FLOATING POINT DUPLICATE

(f1 f2 --- f1 f2 f2)

Duplicates the top floating point number on the stack.

EXAMPLE :

```
1.  2.  FDUP  F.  F.  F.  <CR>  2  2  1 OK
```

F2DUP : FLOATING POINT DOUBLE DUPLICATE

(f1 f2 --- f1 f1 f2 f 2)

Duplicates the top two floating point numbers on the stack.

EXAMPLE :

```
1.  2.  F2DUP  F.  F.  F.  F.  <CR>  2  2  1  1 OK
```

FSWAP : FLOATING POINT SWAP

(f1 f2 --- f2 f1)

Swaps the two top floating point numbers on the stack.

EXAMPLE :

```
1.  2.  FSWAP  F.  F.  <CR>  1  2 OK
```

FOVER : FLOATING POINT OVER

(f1 f2 --- f1 f2 f1)

Leaves a copy of the second floating point number on the top of the stack.

EXAMPLE :

```
1.  2.  FOVER  F.  F.  F.  <CR>  1  1  2 OK
```

FPICK : FLOATING POINT PICK

(n --- f)

Returns the contents of the nth floating point value on the stack.

EXAMPLE :

```
1.  2.  3.  4.  2  FPICK  F.  F.  F.  F.  F.  <CR>
3  4  3  2  1 OK
```

FROLL : FLOATING POINT ROLL

(n --- f)

Rotates the nth stack value to the top of the stack. All other floating point numbers are moved down to fill the vacated position.

FROT : FLOATING POINT ROTATE

(f1 f2 f3 --- f2 f3 f1)

Rotates the third value on the stack to the top of the stack.

EXAMPLE :

```
1.  2.  3.  FROT  F.  F.  F.  <CR>  1  3  2 OK
```

?FDEPTH : FLOATING POINT ?DEPTH

(n ---)

If depth of stack in floating point numbers is less than n, before n was entered, ABORT" is called printing EMPTY STACK.

EXAMPLE :

```
123.456  <CR>

3  ?FDEPTH  <CR>
^EMPTY STACK
```

FDEPTH : FLOATING POINT DEPTH

(--- n)

Returns the depth of the stack in floating point numbers.

EXAMPLE :

```
123.456  45.001  FDEPTH  .  <CR>  2 OK
```

D->F : DOUBLE NUMBER CONVERSION

(d --- f)

Converts a double number on the stack to a floating point number.

EXAMPLE :

```
FLP-INIT

123.456  D->F  F.  <CR>
```

Result is 123456. being displayed.

F->D : FLOATING NUMBER CONVERSION

(f --- d)

Converts the integer portion of a floating point number on the stack to a double number. If the number is less than zero, and contains a decimal fraction, then the number will be rounded down to the next whole integer. In the interest of speed of execution, this word does not check the floating number for being within the range of a double number (+-2147483647). It is left to the user to be aware of range.

EXAMPLE 1 :

```
FLP-INIT

123.456  F->D  D.  <CR>  123 OK
```

EXAMPLE 2 :

```
-123.456  F->D  D.  <CR>  -124 OK
```

F->S : FLOATING NUMBER CONVERSION

(f --- n)

Converts the integer portion of a floating point number between -32767 and 32767 to a single length integer number. If the number is less than zero, and the number contains a decimal fraction, then the number will be rounded down to the next whole integer.

EXAMPLE 1 :

```
FLP-INIT

32000  F->S  .  <CR>
```

Results in 32000 being displayed.

EXAMPLE 2 :

```
-32000  F->S  .  <CR>
```

Results in -32000 being displayed.

EXAMPLE 3 :

```
123.456  F->S  .  <CR>
```

Results in 123 being displayed.

EXAMPLE 4 :

```
-123.456  F->S  .  <CR>
```

Results in -124 being displayed.

S->F : SINGLE NUMBER CONVERSION

(n --- f)

Converts a single length number to a floating point number.

EXAMPLE :

```
FLP-INIT
132  S->F  F.   <CR>
```

Results in 132. being displayed.

$->F : CONVERT STRING TO FLOATING POINT NUMBER

(addr --- f)

Converts a numerical string at address addr to its floating point equivalent, and places the result on the stack.

EXAMPLE :

```
FLP-INIT
20  $VARIABLE  STR1  <CR>
STR1  $CLR  <CR>
```

```
STR1  " 1.23401E+24"  $CONCAT  <CR>
STR1  $->F  F.  <CR>  1.23401E+24 OK
```

Note: if a string has an exponent, such as 1.01E-05, then the exponent must be two numbers long, e.g., If the string value equals 1.009E-9, then the string must read: 1.009E-09. In other words, for our example, -09, not just -9, must be used as the exponent within the string. Also, the mantissa of the number must contain a decimal point, e.g., If the string value is 1E+24, the string must read 1.0E+24. The reason for this is the decimal point signals the number handling routine of SUPER-FORTH to treat the mantissa as a floating point number.

F->$: CONVERT FLOATING POINT NUMBER TO STRING

(f --- addr)

This word converts a floating point number on the stack to a string variable, places it in the PAD and leaves the address of the PAD on the stack.

EXAMPLE :

```
123.456  F->$  <CR>
$.  <CR>  123.456 OK
```

<FNUM> : FLOATING POINT NUMBER CONVERSION ROUTINE

(ADDR --- f)

This routine replaces the standard system number conversion routine (<NUMBER>) when FLP-INIT is invoked. This allows standard single number input, and automatic double number to floating point number conversion, so that words may be defined while in floating point mode.

Floating point numbers are 6 bytes long (3 cells) and are arranged on the stack as follows:

BYTE1 FLOATING POINT EXPONENT

BYTE2 SIGN BYTE FOR THE MANTISSA

BYTES 3-6 THE MANTISSA

In this way, a full four bytes are available for the mantissa, providing greater accuracy and speed.

Results in 32000 being displayed.

EXAMPLE 2 :

```
-32000  F->S  .  <CR>
```

Results in -32000 being displayed.

EXAMPLE 3 :

```
123.456  F->S  .  <CR>
```

Results in 123 being displayed.

EXAMPLE 4 :

```
-123.456  F->S  .  <CR>
```

Results in -124 being displayed.

S->F : SINGLE NUMBER CONVERSION

(n --- f)

Converts a single length number to a floating point number.

EXAMPLE :

```
FLP-INIT

132  S->F  F.   <CR>
```

Results in 132. being displayed.

$->F : CONVERT STRING TO FLOATING POINT NUMBER

(addr --- f)

Converts a numerical string at address addr to its floating point equivalent, and places the result on the stack.

EXAMPLE :

```
FLP-INIT

20  $VARIABLE  STR1  <CR>
STR1  $CLR  <CR>
```

```
STR1  " 1.23401E+24"  $CONCAT  <CR>
STR1  $->F  F.  <CR>  1.23401E+24 OK
```

Note: if a string has an exponent, such as 1.01E-05, then the exponent must be two numbers long, e.g., If the string value equals 1.009E-9, then the string must read: 1.009E-09. In other words, for our example, -09, not just -9, must be used as the exponent within the string. Also, the mantissa of the number must contain a decimal point, e.g., If the string value is 1E+24, the string must read 1.0E+24. The reason for this is the decimal point signals the number handling routine of SUPER-FORTH to treat the mantissa as a floating point number.

F->$: CONVERT FLOATING POINT NUMBER TO STRING

(f --- addr)

This word converts a floating point number on the stack to a string variable, places it in the PAD and leaves the address of the PAD on the stack.

EXAMPLE :

```
123.456  F->$  <CR>
$.  <CR>  123.456 OK
```

<FNUM> : FLOATING POINT NUMBER CONVERSION ROUTINE

(ADDR --- f)

This routine replaces the standard system number conversion routine (<NUMBER>) when FLP-INIT is invoked. This allows standard single number input, and automatic double number to floating point number conversion, so that words may be defined while in floating point mode.

Floating point numbers are 6 bytes long (3 cells) and are arranged on the stack as follows:

BYTE1 FLOATING POINT EXPONENT

BYTE2 SIGN BYTE FOR THE MANTISSA

BYTES 3-6 THE MANTISSA

In this way, a full four bytes are available for the mantissa, providing greater accuracy and speed.

FLITERAL : FLOATING POINT LITERAL

(f ---)

If compiling, then compile the stack value f as a 48-bit literal, which when later executed will leave f on the stack.

EXAMPLE :

```
FLP-INIT

: TEST [ 123.456 ] FLITERAL ; <CR>

TEST F. <CR> 123.456 OK
```

MATRIX WORD SET

for SUPER-FORTH 64™

By Bruce Jordan

LOADING INSTRUCTIONS

To load the MATRIX WORD SET,

1. Load and run SUPER-FORTH.

2. Insert the SUPER-FORTH EXTENDED MATH-I/O UTILITY DISK into your disk drive.

3. Type 60 77 THRU

To load the LATTICE WORD SET, type:

```
78  82  THRU
```

USING MATRIX WORDS

The following is a few examples on using the MATRIX word set.

First, let's try creating a 2-dimensional matrix of 3 rows and 3 columns.

EXAMPLE :

```
3  3  MDIM  M1
```

We now have a matrix called M1.

Next, we might want to put some values into our matrix. We do this by using the word MAT!:

EXAMPLE :

```
1  0  0  M1  MAT!  <CR>  OK
2  1  0  M1  MAT!  <CR>  OK
3  2  0  M1  MAT!  <CR>  OK
4  0  1  M1  MAT!  <CR>  OK
5  1  1  M1  MAT!  <CR>  OK
6  2  1  M1  MAT!  <CR>  OK
7  0  2  M1  MAT!  <CR>  OK
8  1  2  M1  MAT!  <CR>  OK
9  2  2  M1  MAT!  <CR>  OK    ( WHEW! )
```

Now that wasn't so easy was it. Notice that first number was the value you were storing in the matrix element, and the second and

third numbers were the X and Y coordinates of the elements. Also notice that the X and Y coordinates (row and column location) range from 0 to 1 minus the number of rows and columns.

Now, we can use MAT? to print out what we have in our matrix.

EXAMPLE :

```
M1  MAT?  <CR>  ( PRINT OUT MATRIX )
1  2  3
4  5  6
7  8  9
OK
```

We can fetch any element within M1 by using MAT@. Let's try fetching the value at coordinates X=0, Y=1.

```
0  1  MAT@  .  <CR>  4  OK
```

Next, let's create another matrix:

EXAMPLE :

```
3  3  MDIM  M2  <CR> OK
```

This time, we want all of the values in M2 to be the same, so we'll use MFILL:

EXAMPLE :

```
5  M2  MFILL  <CR> OK   ( FILL M2 WITH 5's )
```

If we print M2, we get...

```
M2  MAT?  <CR>
5  5  5
5  5  5
5  5  5
OK
```

Next, we'll add the contents of matrix M2 to M1. This will result in five being added to every element of M1. However, we must first define a third matrix to hold the result of the addition:

```
3  3  M3  MDIM  <CR>  OK
```

Since we are using M3 only to hold the result of the addition of M1 and M2, we really don't care what is initially stored in M3. Therefore, we won't store anything in M3 for now.

Now, we can perform our matrix addition.

EXAMPLE :

```
M3  M2  M1   MAT+  <CR> OK
```

If we print out M3, we see that we have:

```
M3  MAT?  <CR>
6  7  8
9  10  11
12  13  14
OK
```

Note that this result is:

```
1  2  3          5  5  5        6  7  8
4  5  6    +     5  5  5    =   9  10  11
7  8  9          5  5  5        12  13  14

  M1       +       M2      =         M3
```

As our final example, we will transpose matrix M1. This means that we will swap rows for columns and visa versa. The result of this will be stored in matrix M2.

EXAMPLE :

```
M1  M2  MTRN  <CR>  OK  ( TRANSPOSE M1. )
```

If we print out M2 and M1, we get:

```
M2  MAT?  <CR>
1  4  7
2  5  8
3  6  9
OK

M1  MAT?  <CR>
1  2  3
4  5  6
7  8  9
OK
```

Notice that the rows of M1 have become the columns of M2, and the columns of M1 have become the rows of M2.

ERROR CHECKING

There are a number of error checking words used by the various MATRIX and LATTICE words to make sure that operations performed on matrices stay within bounds, and do not over-write other areas in memory. These words are characterized by a question mark at the beginning of their names. These words may be removed from the definitions of the MATRIX and LATTICE words to increase speed of execution. However, if the error checking words are removed, extreme care must be exercised.

LIST OF ERROR CHECKING WORDS

?=, ?TRN, ?MAT123, ?MAT23, ?123+, ?L=

NOTE : MAT? IS NOT AN ERROR CHECKING WORD.

RE-DIMENSIONING MATRICES

Certain matrix words will automatically re-dimension a matrix. Therefore, the user should make certain to keep aware of any changes in the size of the dimensions of his or her matrices. For instance, if a matrix was dimensioned as 5 rows and 4 columns, and if a particular operation re-dimensions the matrix to 2 rows and 2 columns, then it no longer makes sense to refer to the coordinates of any element greater than X=1, Y=1 within the re-dimensioned matrix.

LIST OF WORDS THAT CAN RE-DIMENSION A MATRIX

MAT*, MAT/, MAT=, MAT+, MAT-, MTRN

MATRIX AND LATTICE STRUCTURE

This section is intended only as a reference work for the convenience of the advanced hacker. It is not necessary for the

beginner to read this section in order to use the MATRIX word sets.

A two-dimensional matrix is set up in memory with the following structure:

X coordinate first, then Y coordinate, and finally, the body of the matrix. The matrix body is arranged in memory such that the element are stored in memory from left to right and top to bottom.

EXAMPLE :

If we have a matrix...

```
  1   23   4   5
  0   13  17   2
 55   14  19   3
  6    2   8  12
```

Then the matrix resides in memory like this:

X Y 1 23 4 5 0 13 17 2 55 14 19 3 6 2 8 12

Where X=4 and Y=4.

Also, remember that each number in the matrix resides in a CELL of memory (2 bytes). Therefore, the above example takes up 4 bytes for the dimension sizes and 32 bytes for the body of the matrix. A total of 36 bytes for the entire matrix. The rule for matrix size is:

2*X*Y+4=MATRIX MEMORY USAGE.

The structure of a LATTICE is a little different. In a LATTICE, the first two bytes hold the number of dimensions, followed by a series of 2-byte cells containing the size of the particular dimensions, and finally, the body of the LATTICE. The storing of the LATTICE body in memory is rather difficult to visualize. The body of the LATTICE is stored from left to right and top to bottom as with a MATRIX, but then the LATTICE is stored from front layer to back layer. If the LATTICE has more than three dimensions, then these dimensions are also taken into account in the positioning of the elements in memory. It can get complicated really fast! Fortunately, the LATTICE words have been written such that this storing process is taken care of for the user.

EXAMPLE :

A four dimensional LATTICE is structured like this:

ND X Y Z T 0000 0001 0002 0003 ....

Where

ND=the number of dimensions

X, Y, Z, and T=the size of the dimensions

0001, 0002, 0003,...000N=coordinates of the elements.

The equation for finding the offset into an n-dimensional matrix for a particular element is:

OFFSET=DXy+DXDYz+DXDYDZt+...+DXDYDZDT...(DN-1)n+x+2ND+2

Where

DX, DY, DZ, DT, DN=size of dimensions

x, y, z, t, n=coordinates within the LATTICE, and

ND=number of dimensions.

The rule for determining the size of a LATTICE in terms of memory usage is:

2*X*Y*Z*...*N+2*ND+2=LATTICE MEMORY USAGE

Where

X, Y, Z, N=size of dimensions, and

ND=number of dimensions

2-DIMENSIONAL MATRIX WORD SET

Whenever a word uses [NAME], the word is a CREATING word and requires that a name be typed in after the creating word and before the carriage return is typed. [NAME] is the stack notation for this name. [NAME] is not left on the stack, as the stack notation might suggest.

EXAMPLE :

```
CREATE  SPOT  <CR>  OK  ( SPOT=[NAME] )
```

This example creates an entry in the system dictionary called SPOT. Notice that the name of the area was typed in after the FORTH word CREATE, and not before it. The stack notation for CREATE would be as follows:

```
( --- [NAME] )
```

The following is a list of the 2-dimensional matrix words and their usage.

MDIM : CREATE A 2 DIMENSIONAL MATRIX.

```
( X Y --- [NAME] )
```

Creates a two dimensional matrix with dimensions (COLUMN and ROW) X and Y.

EXAMPLE :

```
5  4  MDIM  LEDGER  <CR> OK
```

Creates a two dimensional matrix with size X=5 (COLUMNS) and Y=4 (ROWS).

MELEMENT : RETURNS ADDRESS OF ELEMENT IN MATRIX.

```
( X Y ADDR --- ADDR )
```

Returns the address of an element of a two dimensional matrix, where X and Y are the coordinates of the element. Note that the

coordinates range from zero to the size of the particular dimension minus one, eg: If a matrix was dimensioned as X=5 and Y=4, then the coordinates of the first element of the matrix would be 0 and 0, and the coordinates of the last element would be 4 (5-1) and 3 (4-1).

EXAMPLE :

```
1  3  LEDGER  MELEMENT  <CR> OK
```

Returns the address of the element of LEDGER in the second column and fourth row.

?= : DIMENSION TEST

(M1 M2 --- M1 M2)

This word is a test to determine whether or not matrix M2 is larger than or equal to M1. If no, then execution is aborted, and error message "MATRIX MISMATCH" is printed. This word is used in error checking in MAT=.

MAT= : SET MATRIX2 EQUAL TO MATRIX1

(M1 M2 ---)

Sets the contents of matrix M2 equal to the contents of matrix M1. The number of rows and columns in matrix M2 must be equal to or greater than the number of rows and columns of matrix M1. Care should be exercised in the use of this word, because matrix M2 will be re-dimensioned to the dimensions of matrix M1.

EXAMPLE :

```
LEDGER1  LEDGER2  MAT=  <CR> OK
```

Sets the contents of LEDGER2 equal to the contents of LEDGER1.

MAT@ : FETCH A VALUE FROM A MATRIX

(X Y ADDR ---)

Fetches a value from an element of a matrix, and places it on the stack.

EXAMPLE :

```
0  0  LEDGER  MAT@  <CR> OK
```

Fetches the value stored in the first element of matrix LEDGER.

MAT! : STORES A VALUE INTO MATRIX

(N X Y ADDR ---)

The value n is stored in the element of a matrix at location X, Y.

EXAMPLE :

```
123  0  0   LEDGER  MAT!  <CR> OK
```

Stores the value 123 into the first element of matrix LEDGER.

GRAB : COPY FOURTH STACK VALUE

(N1 N2 N3 N4 --- N1 N2 N3 N4 N1)

Leaves a copy of the fourth value on the stack on the top of the stack.

EXAMPLE :

```
1  2  3  4  GRAB  <CR> OK
```

Leaves 1 2 3 4 1 on the stack.

MFILL : FILL A MATRIX

(N ADDR ---)

Fills all elements of a matrix with the value N.

EXAMPLE :

```
23  LEDGER  MFILL  <CR> OK
```

Fills all elements of matrix LEDGER with 23.

?TRN : DIMENSION TEST

(M1 M2 --- M1 M2)

Test to see if matrix transposition is legal. This word tests for matrix M2 being of same absolute dimensions (ROWS*COLUMNS) as matrix M1, if not execution is halted and the error message "MATRIX MISMATCH" is printed.

TRNSET : SET DIMENSIONS

(M1 M2 --- M1 M2)

This word re-dimensions matrix M2 such that the number of rows of matrix M2 is equal to the number of columns of matrix M1, and the number of columns of matrix M2 is equal to number of rows of matrix M1. This word is used by MTRN.

MTRN : TRANSPOSE A MATRIX

(M1 M2 ---)

This word performs a transposition on matrix M1 (exchanges row elements with column elements), and leaves the result in matrix M2. The size of matrix M2 must be greater than or equal to matrix M1. This word re-dimensions matrix M2 to the dimensions of matrix M1.

EXAMPLE :

LEDGER1 LEDGER2 MTRN <CR> OK

LEDGER2 becomes the transposition of LEDGER1.

SCALAR+ : ADD VALUE TO MATRIX

(N ADDR ---)

This word adds a scalar (numeric) value N to all elements of a matrix.

EXAMPLE :

65 LEDGER SCALAR+ <CR> OK

65 is added to the contents of all elements of LEDGER.

SCALAR* : MULTIPLY MATRIX BY VALUE

(N ADDR ---)

This word multiplies all elements in a matrix by a scalar (numeric) value N.

EXAMPLE :

3 LEDGER SCALAR* <CR> OK

The contents of all elements of LEDGER are multiplied by 3.

SCALAR- : SUBTRACT FROM MATRIX

(N ADDR ---)

Subtracts a scalar value N from all elements in a matrix.

EXAMPLE :

2 LEDGER SCALAR- <CR> OK

2 is subtracted from the contents of all elements of LEDGER.

SCALAR/ : DIVIDE MATRIX BY VALUE

(N ADDR ---)

Divides all elements in matrix by a scalar (numeric) value N.

EXAMPLE :

2 LEDGER SCALAR/ <CR> OK

The contents ofall elements of LEDGER are divided by 2

?MAT123 : DIMENSION TEST

(M1 M2 M3 --- M1 M2 M3)

Matrix test to see if legal to perform matrix multiplication. Checks if number of rows of matrix M3 is equal to number of rows of matrix M1, and number of columns of matrix M3 is equal to number of columns of matrix M2. If not, execution is halted, and the error messages "COLUMN MISMATCH" or "ROW MISMATCH" will be

printed.

?MAT23 : DIMENSION TEST

(M1 M2 --- M1 M2)

Matrix test to see if matrix multiplication is legal. Tests if number of rows of matrix M2 is equal to number of columns of matrix M1. If not, execution is halted, and the error message "COLUMN/ROW MISMATCH" will be printed.

K : RETURN LOOP INDEX

(--- N)

Returns index of third outer loop of three nested loops.

EXAMPLE :

```
:  TEST
   10  0  DO
        3  0  DO
                 3  0  DO
                          I  J  K  .  .  .  CR
                       LOOP
              LOOP
         LOOP ;
```

Will print the indexes of the DO-loops

MAT* : MATRIX MULTIPLICATION

(M1 M2 M3 ---)

Multiplies matrix M3 by matrix M2, the result is left in matrix M1. Matrix M1 is re-dimensioned to the number of rows of M2 and the number of columns of M3. Matrix M1 must have a number of rows greater than or equal to matrix M2, and matrix M1 must have a number of columns greater than or equal to matrix M3. Also, matrix M3 must have a number of rows equal to the number of columns of matrix M2.

EXAMPLE :

```
LEDGER1  LEDGER2  LEDGER3  MAT*  <CR> OK
```

Multiplies (using matrix multiplication) LEDGER3 by LEDGER2, and leaves the result in LEDGER1.

MAT/ : MATRIX DIVISION

(M1 M2 M3 ---)

Divides matrix M3 by matrix M2, the result is left in matrix M3. Matrix M3 is re-dimensioned to the number of rows of M2 and the number of columns of M3. Matrix M1 must have a number of rows greater than or equal to matrix M2, and matrix M1 must have a number of columns greater than or equal to matrix M3. Also, matrix M3 must have a number of rows equal to the number of columns of matrix M2.

EXAMPLE :

LEDGER1 LEDGER2 LEDGER3 MAT/ <CR> OK

Divides (using matrix multiplication) LEDGER3 by LEDGER2, and leaves the result in LEDGER1.

?123+ : DIMENSION TEST

(M1 M2 M3 --- M1 M3 M2)

Tests to see if matrix addition is legal. This word checks if matrix M2 is equal to matrix M3, and if matrix M1 is greater than or equal to matrix M2. If not, then execution is halted, and one of three error messages, "COLUMN MISMATCH", "ROW MISMATCH" or "MATRIX MISMATCH" will be printed.

MAT+ : MATRIX ADDITION

(M1 M2 M3 ---)

Adds each element of matrix M2 to each corresponding element of matrix M3. The result is left in matrix M1. The number of rows and columns of matrix M2 must be equal to the number of rows and columns of matrix M3. Also, the number of rows, and columns of matrix M1 must be greater than or equal to the number of rows and columns of M2 or M3.

EXAMPLE :

LEDGER1 LEDGER2 LEDGER3 MAT+ <CR> OK

Adds the contents of each element of LEDGER2 to the contents of the corresponding element in LEDGER3, leaving the result in the corresponding element of LEDGER1.

MAT- : MATRIX SUBTRACTION

(M1 M2 M3 ---)

Subtracts each element of matrix M2 from each corresponding element of matrix M3. The result is left in matrix M1. The number of rows and columns of matrix M2 must be equal to the number of rows and columns of matrix M3. Also, the number of rows and columns of matrix M1 must be greater than or equal to the number of rows and columns of M2 or M3.

EXAMPLE :

```
LEDGER1  LEDGER2  LEDGER3  MAT-  <CR> OK
```

Subtracts the contents of each element of LEDGER2 from the contents of the corresponding element in LEDGER3, leaving the result in the corresponding element of LEDGER1.

MAT? : PRINT A MATRIX

(ADDR ---)

This word prints the contents of a matrix on the screen.

EXAMPLE :

```
3  3  MDIM  LEDGER  <CR>  OK( CREATE A MATRIX )

5  LEDGER  MFILL  <CR>  OK( FILL THE MATRIX WITH 5's )

LEDGER  MAT?  <CR>  ( PRINT MATRIX )
5   5   5
5   5   5
5   5   5
OK
```

MULTI-DIMENSIONAL MATRIX WORD SET

These words are used to create and manipulate matrices of three or greater dimensions. To distinguish them from the two dimensional matrix words, a matrix of three dimensions or more will be referred to (though not accurately) as a **LATTICE.**

LDIM· : CREATE AN N DIMENSIONAL LATTICE

(DX DY DZ...DN N --- [NAME])

This word creates an N dimensional (3 or more dimensions) lattice, where DX through DN are the sizes of the particular dimensions, and N is the number of dimensions.

EXAMPLE :

```
3  5  6  7  4  DIM  TIMECUBE  <CR> OK
```

The above example will create a 4-dimensional lattice, with X=3, Y=5, Z=6 and fourth-dimension equal to 7 (T=7).

LELEMENT : RETURN ADDRESS OF ELEMENT OF A LATTICE

(N...Z Y X ADDR --- ADDR)

This word returns the address of an element of an N-dimensional lattice, where X through N are the coordinates within the lattice of the particular element.

NOTE : Coordinates range from zero to one minus the size of the dimensions. In other words, if the lattice was dimensioned at X=3, Y=3 and Z=3, the coordinates of the elements would range from (0, 0, 0) to (2, 2, 2).

EXAMPLE :

The lattice TIMECUBE has been dimensioned to X=3, Y=5, Z=6 and T=7. Therefore, the coordinates of the last element of TIMECUBE would be: X=2, Y=4, Z=5 and T=6.

```
6  5  4  2  TIMECUBE  LELEMENT  <CR> OK
```

Returns the address of the last element of lattice TIMECUBE.

LFILL : FILL A LATTICE

(N ADDR ---)

Fills all elements of an n-dimensional lattice with value N.

EXAMPLE :

234 TIMECUBE LFILL <CR> OK

Fills all elements of lattice TIMECUBE with 234.

?L= : CHECK LATTICE EQUALITY

(L1 L2 --- L1 L2)

Checks to see if lattice L1 is equal to lattice L2, if not then execution is halted, and error message: "LATTICE MISMATCH" is printed. This word is used in error checking by LAT=.

LAT= : SET TWO LATTICES EQUAL

(L1 L2 ---)

Copies the contents of lattice L1 into lattice L2. The two lattices must be of equal number of dimensions and equal size of dimensions.

EXAMPLE :

TIMECUBE1 TIMECUBE2 LAT= <CR> OK

Copies contents of TIMECUBE1 into TIMECUBE2.

LAT@ : FETCH ELEMENT FROM LATTICE

(N...Z Y X ADDR --- VAL)

Returns the contents of an element of an n-dimensional lattice.

EXAMPLE :

0 0 0 0 TIMECUBE LAT@ <CR> OK

Returns the contents of the first element of the lattice TIMECUBE.

LAT! : STORE VALUE IN LATTICE ELEMENT

(VAL N...Z Y X ADDR ---)

Stores a value into an element of an n-dimensional lattice.

EXAMPLE :

```
123  0  0  0  0  TIMECUBE  LAT!  <CR> OK
```

Stores 123 in first element of the lattice TIMECUBE.

ALGEBRAIC EXPRESSION EVALUATOR PACKAGE

for SUPER-FORTH 64™

Program written by Michael Stolowitz

Adapted by Bruce Jordon 1984

®PARSEC RESEARCH

LOADING INSTRUCTIONS

To load the ALGEBRAIC EXPRESSION EVALUATOR,

1. Load and run SUPER-FORTH.

2. Insert the SUPER-FORTH EXPANDED MATH-I/O UTILITY DISK into your disk drive.

3. Type 83 LOAD

As the ALGEBRAIC EVALUATOR loads, will will see several "ISN´T UNIQUE" messages printed on the screen. Don´t worry about this. The SUPER-FORTH arithmetic word set is simply being re-defined.

After loading the ALGEBRAIC EVALUATOR and the Extended Floating Point Math Package (Both contained on this disk.), you may set the ALGEBRAIC EVALUATOR to work with floating point numbers. For instructions on how to do this, see the section on using the ALGEBRAIC EVALUATOR with floating point numbers.

ALGEBRAIC NOTATION WITH SUPER-FORTH

The Algebraic Expression Evaluator, is a FORTH extension that allows the inputing of functions in Algebraic form, rather than the standard FORTH Reverse Notation form.

Before Algebraic Notation arithmetic can be performed, the standard FORTH math words (* / + - =...) must be re-defined. Consider the following expression:

```
3 * 5 + 2
```

We would solve this problem, by first multiplying 3 by 5, then adding 2 to the result, would arrive at 17 as our answer. Note that if we first added 2 to 5, and then multiplied by 3, we would get 21 as our answer. However, In this case, we say that multiplication has a higher precedence than addition, and is usually performed first. Therefore, 17 is the correct answer. As another example, consider:

```
2 * 3 / ( 4 - 1 )
```

In this example, 2 is multiplied by 3, then the result is divided by 4-1. The parentheses set off the difference of 4 and 1 so that they're considered as one number. Parentheses say: "Do what's inside me before doing anything else!" As a result, parentheses set another level of precedence.

To assign a PRECEDENCE VALUE to a mathematical operator, we use the word INFIX. This PRECEDENCE VALUE is used to determine the order in which operations will be performed within an algebraic expression.

EXAMPLE 1 :

```
7  INFIX  *  *  <CR> OK
```

This sets the precedence value of multiplication to 7.

EXAMPLE 2 :

```
6  INFIX  +  +  <CR>  OK
```

This sets the precedence value of addition to 6. Note that this is a lower precedence value than multiplication, and if encountered within an algebraic expression, addition will be performed after multiplication.

Once the precedence value of a FORTH math word has been defined, it may be used within an algebraic expression. The precedence values for many of the standard FORTH words have already been set. However, these values may be changed to fit the particular needs of the user.

Next, a way is needed to signal the operating system that the following mathematical expression should be evaluated in algebraic form. We have a special FORTH word for this: A[(Pronounced A BRACKET); and the word for ending algebraic evaluation is]A (Pronounced BRACKET A), of course.

EXAMPLE 1 :

```
A[ 3 + 4 - 5 * ( 8 / 2 ) ]A  <CR> OK
```

Leaves -13 on the stack.

EXAMPLE 2 :

```
A[ ( 3 * 5 ) - ( 2 * 3 ) ]A  <CR> OK
```

Leaves 9 on the stack. Notice that the parts of the expression enclosed in parentheses are calculated first.

USING FLOATING POINT WITH ALGEBRAIC EVALUATOR

Floating point math functions of the Extended Floating Point Math Package (contained on this disk) may be used with the Algebraic Expression Evaluator. Keep in mind that the precedence of the floating point words must be set. loading screens 86-87 from the SUPER-FORTH Extended Math disk will automatically set the precedence of the floating point words.

To use the ALGEBRAIC EVALUATOR with floating point numbers,

1. Load and run SUPER-FORTH.

2. Insert the SUPER-FORTH Extended Math disk into your disk drive.

3. Load the ALGEBRAIC EVALUATOR by typing:

   83 LOAD

4. Load the Extended Floating Point Math Package by typing:

   2 LOAD

5. Finally, load in the re-defined floating point word set by typing:

   86 LOAD

The floating point words may now be used with the ALGEBRAIC EXPRESSION EVALUATOR.

Keep in mind the change in syntax when using the ALGEBRAIC EVALUATOR. For instance, if in floating point mode, FCOS may be used to obtain the cosine of a number:

EXAMPLE :

```
FLP-INIT  <CR> OK

1.234  FCOS  <CR> OK
```

However, in Algebraic Notation it would be done like this:

```
FLP-INIT  <CR> OK

A[ FCOS 1.234 ]A  <CR> OK
```

As another example, the floating point exponent of a number is entered in standard FORTH Notation like this:

```
FLP-INIT  <CR> OK

4.5  2.0  F^  <CR> OK ( RETURNS 4.5 TO THE SECOND POWER.)
```

However, in Algebraic Notation we would use the form:

```
FLP-INIT  <CR> OK

A[ 4.5 F^ 2.0 ]A  <CR> OK ( RETURNS 4.5 TO THE SECOND)
                          ( POWER. )
```

CONSTANTS WITH ALGEBRAIC EVALUATOR

Besides single numbers and floating point numbers, FORTH constants may also be used with the Algebraic Expression Evaluator. This allows the creation of code that resembles the familiar letter-number look of algebra.

EXAMPLE :

```
 5  CONSTANT  X  <CR> OK
 6  CONSTANT  Y  <CR> OK
11  CONSTANT  Z  <CR> OK

A[ 2 * X + 3 * Y + Z ]A  <CR> OK
```

Leaves 39 on the stack.

THINGS TO REMEMBER

1. Math words must have a precedence value assigned to them by INFIX before they can be used with the Expression

Evaluator.

2. Algebraic Expressions must begin with A[and end with]A.

3. Constants may be used, as well as numbers.

4. The Algebraic Expression Evaluator supports the use of many of the floating point words.

5. As usual, you must be in floating point mode to use floating point numbers within algebraic expressions.

6. Parentheses may be used to separate operations.

7. After being INFIXed, some of the FORTH math words may react strangely if not used within algebraic Mode. For this reason, it's not a good idea to go back and forth between algebraic and standard modes.

LIST OF SUPPORTED MATH OPERATION WORDS

The following is a list of SUPER-FORTH math words, and their precedence values. These words can be used within an algebraic expression.

PRECEDENCE	WORD
8	FSQR
8	F^
8	FE^X
8	FLOG
8	FLOGX
8	FCOS
8	FSIN
8	FTAN
8	FACOS
8	FASIN
8	FATN
7	F*
7	F/
7	*
7	/
6	F+
6	F-
6	+

6	-
5	F<
5	F>
5	F=
5	<
5	>
5	=
4	NOT
3	FAND
3	AND
2	FOR
2	OR

Index