

Supplement 2
**ANS Forth
Floating Point**



**Triangle Digital
Support Ltd**

TDS2020F ANS FORTH FLOATING POINT

DESCRIPTION

TDS-FLOAT-ANS is a fully featured floating point handling package for the TDS2020F single board computer. It contains all of the American National Standard Forth floating-point and floating-point extension words, plus many other useful functions.

The software is supplied in source code. Apart from normal calculator operations, a full range of trigonometrical, logarithm, exponential and hyperbolic functions is included with 8 to 10 digits accuracy. The code is not re-entrant so floating point words should only be used in one task of a multi-tasking system.

It is available as a password-protected file ansfp.zip file in the 'downloads' section of www.TriangleDigital.com or on the Triangle Digital Services CD. Purchase of a licence for this software allows you to use the compiled code in an unlimited way, but not to divulge the source code to any person or corporate body.

FLOATING POINT FAST START GUIDE

TO INPUT NUMBERS

FLOAT-ON enables the Forth interpreter to accept numbers in scientific format (it should be in the word executed at power-up in a final system ROM). This is how floating point literals should be entered—for example to store and print 16-bit, 32-bit and floating point representations of the number ten:

```
: TEST1 ( - ) 10 . 10. D. 1.0E1 F. ;
```

On execution the word **TEST1** prints:

```
10 10 10.000000
```

Strings can be converted to floating point numbers with **>FLOAT**. Note that as well as the resulting floating-point number it also leaves a flag to indicate whether the conversion was successful. For instance:

```
: TEST2 ( - ) S" 1234.5678" >FLOAT DROP F. ;
```

A third (non ANS Forth) way is provided in this package for applications that need to get a number from the operator of the machine you're designing. The word **FINPUT** gets a floating-point number from the user. He can type it with or without

a decimal point, or even in scientific notation, and the result will be correct. For example:

```
: TEST3 ( - r )  
  INPUT" Input a number: " ." The number is " F. ;
```

This is what may happen when executed. Note that if a mistake is made the number is re-requested:

```
TEST  
  Input a number: 123.456 The number is 123.45600  
  Input a number: 13 The number is 13.000000  
  Input a number: 13k.9  
  Input a number: 13.9 The number is 13.900000
```

TO OUTPUT NUMBERS

We already saw that `F.` prints the floating point number on the top of the stack. For instance `1.23456789E3 F.` will print `1234.5679` when executed, also illustrating the rounding to 8 significant digits. Eight is just the default; you can choose any number up to ten. The last two places may not correct, it depends on the arithmetic you are performing. Set the number of digits as follows:

```
5 SET-PRECISION 1234.5678E0 F.
```

This will display `1234.6` (note the correct rounding) and only 5 digits will be used until `SET-PRECISION` is used again.

Both scientific and engineering formatting are also supported:

```
: TEST4 ( - ) 1.2345678E2 FDUP F. FDUP FE. FS. ;
```

When executed this will illustrate the three formats:

```
123.45678 123.45678E0 1.2345678E2
```

All output is via `EMIT` so you can re-vector it as usual to display floating-point numbers on an LCD for example.

TYPE CONVERSION

You can also input integers as 16-bit or 32-bit numbers, followed by a conversion to the floating-point type. For example:

Supplement 2

```
1234 S>F 12345678. D>F
```

In a definition you get faster execution by doing the conversion in the execute rather than compile mode:

```
[ 1234 S>F ] FLITERAL [ 12345678. D>F ] FLITERAL
```

In the reverse direction any fractional part is discarded. E.g.

```
: TEST5 ( - )
1234567.8E0 F>D D. -1234.5678E0 F>S . ;
```

TEST5 will display 1234567 -1234 .

ARITHMETIC & STACK MANIPULATION

This is straightforward but you may need to remember that each floating-point number in this package is represented by three 16-bit entries on the stack. This means that `F0VER` is the same as a hypothetical word `30VER` .

TRIGONOMETRY

You are not restricted to 90 degrees, or even to a full circle. All angles work but you must specify them in radians, not degrees. This package has the non ANS Forth words `DEG>RAD` and `RAD>DEG` to do the conversion. For instance:

```
30 S>F DEG>RAD FSIN F.
```

will display 0.50000000 which is the sine of 30 degrees.

SUMMARIES BY FUNCTION

STACK & ADDRESS MANIPULATION

<code>F!</code>	<code>(r aa -)</code>	Store r at address aa
<code>F@</code>	<code>(aa - r)</code>	Fetch r from address aa
<code>FALIGN</code>	<code>(-)</code>	Align data-space pointer
<code>FALIGNED</code>	<code>(ca - aa)</code>	aa is aligned address \geq ca
<code>FCONSTANT</code>	<code>(r "<spaces>name" -)</code>	Define floating point constant
	<code>name Execution: - r)</code>	Place r on stack
<code>FDEPTH</code>	<code>(- +n)</code>	+n is possible floating-point values on stack
<code>FDROP</code>	<code>(r -)</code>	Remove r from stack
<code>FDUP</code>	<code>(r - r r)</code>	Duplicate r
<code>FINT</code>	<code>(r1 - r2)</code>	Round r1 to integer (towards 0)

ANS Forth Floating Point

FLITERAL	Compilation: (r -) Run-time: (- r)	Compile r Place r on stack
FLOAT+	(aa1 - aa2)	Add size of floating-point number (6) to aa1
FLOATS	(n1 - n2)	n2 is size of n1 floating-point numbers
FOVER	(r1 r2 - r1 r2 r1)	Place a copy of r1 on top of the stack
FROT	(r1 r2 r3 - r2 r3 r1)	Rotate top three floating-point entries
FSWAP	(r1 r2 - r2 r1)	Exchange top two floating-point items
FVARIABLE	("<spaces>name" -) (name Execution: - aa)	Create a definition for name, reserve three bytes of data space (which is not initialised) aa is address of the reserved space

ARITHMETIC

F*	(r1 r2 - r3)	Multiply r1 by r2
F+	(r1 r2 - r3)	Add r1 to r2
F-	(r1 r2 - r3)	Subtract r1 from r2
F/	(r1 r2 - r3)	Divide r1 by r2
F0<	(r - flag)	True if r less than zero
F0=	(r - flag)	True if r equal to zero
F2*	(r1 - r2)	Fast floating point multiply by 2
F2/	(r1 - r2)	Fast floating point divide by 2
F<	(r1 r2 - flag)	True if r1 less than r2
FABS	(r1 - r2)	r2 is absolute value of r1
F=	(r1 r2 - flag)	True if r1 is equal to r2
F>	(r1 r2 - flag)	True if r1 greater than r2
FLOOR	(r1 - r2)	Round r1 to integer (towards minus infinity)
FMAX	(r1 r2 - r3)	r3 is greater of r1 and r2
FMIN	(r1 r2 - r3)	r3 is lesser of r1 and r2
FNEGATE	(r1 - r2)	r2 is negation of r1
FROUND	(r1 - r2)	Round r1 to nearest integer
FSQRT	(r1 - r2)	r2 is square root of r1
F~	(r1 r2 r3 - flag)	True if the difference between r1 & r2 is less than r3

Supplement 2

TRIGONOMETRY

DEG>RAD	(r1 – r2)	r2 is r1 degrees converted to radians
FACOS	(r1 – r2)	r2 is radian angle whose cosine is r1
FASIN	(r1 – r2)	r2 is radian angle whose sine is r1
FATAN	(r1 – r2)	r2 angle whose tangent is r1
FATAN2	(r1 r2 – r3)	r3 is radian angle whose tangent is r1/r2
FCOS	(r1 – r2)	r2 is cosine of the radian angle r1
FSIN	(r1 – r2)	r2 is sine of the radian angle r1
FSINCOS	(r1 – r2 r3)	r2 is the sine and r3 the cos of r1
FTAN	(r1 – r2)	r2 is tangent of the radian angle r1
PI PI/2 PI/4	(– r)	Useful floating point constants
RAD>DEG	(r1 – r2)	r2 is r1 radians converted to degrees

HYPERBOLIC FUNCTIONS

FACOSH	(r1 – r2)	r2 is value whose hyperbolic cosine is r1
FASINH	(r1 – r2)	r2 is value whose hyperbolic sine is r1
FATANH	(r1 – r2)	r2 is value whose hyperbolic tan is r1
FCOSH	(r1 – r2)	r2 is hyperbolic cosine of r1
FSINH	(r1 – r2)	r2 is hyperbolic sine of r1
FTANH	(r1 – r2)	r2 is hyperbolic tangent of r1

EXPONENTIALS & POWERS

FALOG	(r1 – r2)	Raise ten to the power r1
FEXP	(r1 – r2)	Raise e to the power r1
FEXPM1	(r1 – r2)	Raise e to the power r1 and subtract one
F**	(r1 r2 – r3)	Raise r1 to the power r2

LOGARITHMS

FLN	(r1 – r2)	r2 is natural logarithm of r1
FLNP1	(r1 – r2)	r2 is natural logarithm of r1, plus one
FLOG	(r1 – r2)	r2 is base-ten logarithm of r1

INPUT FUNCTIONS

>FLOAT	(ca u – r true false)	Convert string to floating-point
D>F	(d – r)	Convert 32-bit double d to floating-point
FINPUT"	(– r)	Request a floating point input
FLOAT-OFF	(–)	Disable floating point input
FLOAT-ON	(–)	Enable floating point input
S>F	(n – r)	Convert 16-bit single n to floating-point

OUTPUT FUNCTIONS

F>S	(r – n)	n is the integer portion of r
F.	(r –)	Display top floating point number on stack
F>D	(r – d)	Floating-point integer part to double
FE.	(r –)	Display top floating point number on stack using engineering notation
FS.	(r –)	Display top number on stack in scientific notation
PRECISION	(– u)	No. of digits used by F. FE. or FS.
REPRESENT	(r ca u – n flag1 flag2)	Convert mantissa of r to an ASCII string represented as a decimal fraction (with implied decimal point to the left of first digit), and exponent as n, the sign as flag1 and "valid result" as flag2
SET-PRECISION	(u –)	Set number digits used by F. FE. or FS.

IEEE FLOATING POINT REPRESENTATION

DF!	(r aa –)	Store r as 64-bit IEEE double-precision number at aa
DF@	(aa – r)	Fetch 64-bit IEEE double-precision number stored at aa

Supplement 2

DFALIGN	(-)	Align data-space pointer
DFALIGNED	(ca - aa)	aa is aligned address \geq ca
DFLOAT+	(aa1 - aa2)	Add size of 64-bit IEEE double-precision number (8) to aa1
DFLOATS	(n1 - n2)	n2 is size of n1 64-bit IEEE double-precision numbers
SF!	(r aa -)	Store r as a 32-bit IEEE single-precision number at aa
SF@	(aa - r)	Fetch 32-bit IEEE single-precision number stored at aa to stack
SFALIGN	(-)	Align data-space pointer
SFALIGNED	(ca - aa)	aa is aligned address \geq ca
SFLOAT+	(aa1 - aa2)	Add size of a 32-bit IEEE single-precision
SFLOATS	(n1 - n2)	n2 is size of n1 32-bit IEEE single-precision numbers

NUMBER REPRESENTATION

Floating point numbers consist of three 16-bit cells and are arranged on the stack as follows, the range is $\pm 7.0 \times 10^{9863}$.

exp (exponent)	: s : msb : lsb :
m1 (mantissa top word)	: s : msb : 3sb :
m2 (mantissa bottom word)	: 2sb : lsb :

where msb = 'most significant byte' and 's' = sign etc. Floating-point numbers are 'normalised', i.e. for positive numbers bit 14 of m1 is 1.

SYSTEM DOCUMENTATION

ANS FORTH SYSTEM

TDS2020-FLOAT-ANS: providing the Floating-Point word set. Providing the Floating-Point Extensions word set.

IMPLEMENTATION DEFINED OPTIONS

Format and range of floating-point numbers: see above.

Results of `REPRESENT` when float is out of range: does not occur.

Rounding or truncation of floating-point numbers: `F*` `F+` & `F-` round to nearest but mid-way numbers are rounded up. `F/` truncates.

Size of floating-point stack: shares data stack, 118 bytes allowing 39 floating numbers.

Width of floating-point stack: 3 cells, 48 bits.

AMBIGUOUS CONDITIONS

DF@ or DF! is used with an address that is not double-float aligned: ADDR ERR exception.

F@ or F! is used with an address that is not float aligned: ADDR ERR exception. Floating point result out of range (F/): undefined result.

SF@ or SF! is used with an address that is not single-float aligned: ADDR ERR exception.

BASE is not decimal (REPRESENT F. FE. FS.): output is not affected by BASE .

Both arguments equal zero (FATAN2): FP INV ARG exception.

Cosine of argument is zero as an argument for FTAN : FP DIV 0 exception.

d can't be precisely represented as float in D>F : does not occur.

Dividing by zero (F/): FP DIV 0 exception.

Exponent too big for conversion (DF! DF@ SF! SF@): FP RANGE exception.

Float less than one (FACOSH): FP INV ARG exception

Float less than or equal to minus one (FLNP1): FP INV ARG exception.

Float less than or equal to zero (FLN FLOG): FP INV ARG exception.

Float less than zero (FASINH FSQRT): FP INV ARG exception.

Float magnitude greater than one (FACOS FASIN FATANH): FP INV ARG exception.

Integer part of float can't be represented by d in F>D : FP INV ARG exception.

String larger than pictured numeric output area (F. FE. FS.): does not occur with FE. or FS. because of the limit on PRECISION . F. uses scientific format in place of very long fixed-point strings but this is for convenience only and the threshold length, FXLIMIT , can be independent of the size of the pictured numeric output area.

STACK EFFECT ABBREVIATIONS

Symbol	Data type
+n	non-negative 16-bit single-cell number
aa	aligned address
ca	character-aligned address
d	32-bit signed double-cell number (two's complement)
false	false flag (0)
flag	boolean flag. 0 = false, non-zero = true. flag = -1 when true
flag left by a word	
n	16-bit signed single-cell number
true	true flag (-1)
u	16-bit unsigned single-cell number
r	real (floating point) number (6 bytes, 3 words, 48 bits)

Supplement 2

ALPHANUMERIC WORD DEFINITIONS

>FLOAT ANS

ca u – r true | false

An attempt is made to convert the string specified by ca and u to internal floating-point representation. If the string represents a valid floating-point number in the syntax below, its value r and true are returned. If the string does not represent a valid floating-point number only false is returned. A string of blanks is treated as a special case representing zero.

The syntax of a convertible string is:

<significand>[<exponent>]

where:

<significand> := [<sign>]
 { <digits>[.<digits0>] | .<digits> }
<exponent> := <marker><digits0>
<marker> := {<e-form> | <sign-form>}
<e-form> := <e-char>[<sign-form>]
<sign-form> := { + | - }
<e-char>:= { D | d | E | e }

>FLOAT enables programs to read floating-point data in legible ASCII format. Embedded spaces are explicitly forbidden, as are other field separators such as comma or slash. Example string: 1.234e-8

D>F ANS

d – r

r is the floating-point equivalent of 32-bit integer d.

DEG>RAD

r1 – r2

r2 is r1 degrees converted to radians.

DF! ANS

r aa –

Store the floating-point number r as a 64-bit IEEE double-precision number at aa. If the significand of the internal representation of r has more precision than the IEEE double-precision format, it will be rounded using the “round to nearest” rule.

DF@ ANS

aa – r

Fetch the 64-bit IEEE double-precision number stored at aa to the floating-point stack as r in the internal representation. If the IEEE double-precision significand has more precision than the internal representation it will be rounded to the internal representation using the “round to nearest” rule.

DFALIGN ANS

–

If the data-space pointer is not double-float aligned, reserve enough data space to make it so.

DFALIGNED ANS

ca – aa

aa is the first double-float-aligned address greater than or equal to ca..

DFLOAT+ ANS

aa1 – aa2

Add the size in address units of a 64-bit IEEE double-precision number to aa1, giving aa2.

Supplement 2

DFLOATS **ANS**

$$n1 - n2$$

$n2$ is the size in address units of $n1$ 64-bit IEEE double-precision numbers.

F! **ANS**

$$r \text{ aa} -$$

Store r at aa .

F* **ANS**

$$r1 \ r2 - r3$$

Multiply $r1$ by $r2$ giving $r3$.

F** **ANS**

$$r1 \ r2 - r3$$

Raise $r1$ to the power $r2$, giving the product $r3$.

F+ **ANS**

$$r1 \ r2 - r3$$

Add $r1$ to $r2$ giving the sum $r3$.

F- **ANS**

$$r1 \ r2 - r3$$

Subtract $r1$ from $r2$, giving $r3$.

F. **ANS**

r –

Display, with a trailing space, the top number on the floating-point stack using fixed-point notation:

[-]<digits>.<digits0>

For example, `1.23E3 F.` displays `1230`

F/ **ANS**

r1 r2 – r3

Divide r1 by r2, giving the quotient r3.

F0< **ANS**

r – flag

flag is true if and only if r is less than zero.

F0= **ANS**

r – flag

flag is true if and only if r is equal to zero.

F0>

r – flag

flag is true if r is greater than 0.

F2*

r1 – r2

Fast floating point multiply by 2.

Supplement 2

F2/

$$r1 - r2$$

Fast floating point divide by 2.

F< ANS

$$r1 < r2 - \text{flag}$$

flag is true if r1 is less than r2.

F=

$$r1 == r2 - \text{flag}$$

flag is true if r1 is equal to r2.

F>

$$r1 > r2 - \text{flag}$$

flag is true if r1 is greater than r2.

F>D ANS

$$r - d$$

d is the 32-bit signed-integer equivalent of the integer portion of r. The fractional portion of r is discarded.

F>S

$$r - n$$

n is the 16-bit signed-integer equivalent of the integer portion of r. The fractional portion of r is discarded.

F@ ANS

$aa - r$

r is the value stored at aa.

FABS ANS

$r1 - r2$

r2 is the absolute value of r1.

FACOS ANS

$r1 - r2$

r2 is the principal radian angle whose cosine is r1.

FACOSH ANS

$r1 - r2$

r2 is the floating-point value whose hyperbolic cosine is r1.

FALIGN ANS

—

If the data-space pointer is not float aligned, reserve enough data space to make it so.

FALIGNED ANS

$ca - aa$

aa is the first float-aligned address greater than or equal to ca.

FALOG ANS

$r1 - r2$

Raise ten to the power r1, giving r2.

Supplement 2

FASIN **ANS**

$$r1 - r2$$

r2 is the principal radian angle whose sine is r1.

FASINH **ANS**

$$r1 - r2$$

r2 is the floating-point value whose hyperbolic sine is r1.

FATAN **ANS**

$$r1 - r2$$

r2 is the principal radian angle whose tangent is r1.

FATAN2 **ANS**

$$r1 \ r2 - r3$$

r3 is the radian angle whose tangent is r1/r2. `FSINCOS` and `FATAN2` are a complementary pair of operators which convert angles to 2-vectors and vice-versa. They are essential to most geometric and physical applications since they correctly and unambiguously handle this conversion in all cases except null vectors, even when the tangent of the angle would be infinite.

`FSINCOS` returns a unit vector in the direction of the given angle, measured counter-clockwise from the positive X-axis. The order of results on the stack, namely y underneath x, permits the 2-vector data type to be additionally viewed and used as a ratio approximating the tangent of the angle. Thus the phrase `FSINCOS F/` is functionally equivalent to `FTAN`, but is useful over only a limited and discontinuous range of angles, whereas `FSINCOS` and `FATAN2` are useful for all angles. Vectors in general should appear on the stack in this order.

The argument order for `FATAN2` is the same, converting a vector in the conventional representation to a scalar angle. Thus, for all angles, `FSINCOS FATAN2` is an identity within the accuracy of the arithmetic and the argument range of `FSINCOS`. Note that while `FSINCOS` always returns a valid unit vector, `FATAN2` will accept any non-null vector.

FATANH ANS

$r1 - r2$

$r2$ is the floating-point value whose hyperbolic tangent is $r1$.

FCONSTANT ANS

COMPILATION: r “<spaces>name” –

Create a definition for name with the execution semantics defined below.

name is referred to as an “f-constant.”

RUN-TIME: name Execution: – r

Place r on the floating-point stack.

Typical use: r FCONSTANT name

FCOS ANS

$r1 - r2$

$r2$ is the cosine of the radian angle $r1$.

FCOSH ANS

$r1 - r2$

$r2$ is the hyperbolic cosine of $r1$.

FDEPTH ANS

– +n

+n is the current number of possible floating-point values contained on the data stack.

FDROP ANS

r –

Remove r from the floating-point stack.

Supplement 2

FDUP ANS

r – r r

Duplicate r.

FE. ANS

r –

Display, with a trailing space, the top number on the floating-point stack using engineering notation, where the significand is greater than or equal to 1.0 and less than 1000.0 and the exponent is a multiple of three. E.g.
12.345E-6

FERROR

n –

The word `ERROR` in the ANS Forth ROM is revectored to `FERROR` to catch floating-point errors -42, -43 & -46. These are divide by 0 (FP DIV 0), result out of range (FP RANGE) and invalid argument (FP INV ARG) respectively.

FEXP ANS

r1 – r2

Raise e to the power r1, giving r2.

FEXPM1 ANS

r1 – r2

Raise e to the power r1 and subtract one, giving r2. This function allows accurate computation when its arguments are close to zero, and provides a useful base for the standard exponential functions. Hyperbolic functions such as $\cosh(x)$ can be efficiently and accurately implemented using

FEXP_{M1}, accuracy is lost in this function for small values of x if the word FEXP is used.

An important application of this word is in finance; say a loan is repaid at 15% per year; what is the daily rate? On a computer with single precision (six decimal digit) accuracy:

1. Using FLN and FEXP :

FLN of 1.15 = 0.139762, divide by 365 = 3.82910E-4, form the exponent using FEXP = 1.00038, and subtract one (1) and convert to percentage = 0.038%. Thus we only have two digit accuracy.

2. Using FLNP1 and FEXP_{M1} :

FLNP1 of 0.15 = 0.139762, (this is the same value as in the first example, although with the argument closer to zero it may not be so) divide by 365 = 3.82910E-4, form exponent and subtract one (1) using FEXP_{M1} = 3.82983E-4, and convert to percentage = 0.0382983%.

This is full six digit accuracy. The presence of this word allows the hyperbolic functions to be computed with usable accuracy.

FINPUT"

r1 – r2

Request a floating-point input from the terminal or whatever input device ACCEPT is revector to. Use as:

INPUT" Input a number : "

It is similar to BASIC's INPUT and will keep asking if an invalid number is entered.

FINT

r1 – r2

r2 is the floating point integer value of r1, rounded towards 0.

Supplement 2

FLITERAL ANS

COMPILATION: r –

Append the run-time semantics given below to the current definition.

RUN-TIME: – r

Place r on the floating-point stack.

Typical use:

```
: X ... [ ... (r) ]  
      FLITERAL ... ;
```

FLN ANS

r1 – r2

r2 is the natural logarithm of r1.

FLNP1 ANS

r1 – r2

r2 is the natural logarithm of the quantity r1 plus one.

This function allows accurate compilation when its arguments are close to zero, and provides a useful base for the standard logarithmic functions.

See: FEXPM1

FLOAT+ ANS

aa1 – aa2

Add the size in address units of a floating-point number to aa1, giving aa2.

FLOAT-OFF

–

Restore original vectors executed by ?NUMBER and ERROR . This must be done before executing an earlier MARKER word like FP or FP2 . To ensure this FLOAT-OFF is incorporated into redefinitions of marker words FP and FP2.

FLOAT-ON

—

Revector `?NUMBER` to `F?NUMBER` and revector `ERROR` to `FERROR` so that the floating-point input can be recognised. `FLOAT-ON` should be included in the word executed at power-up in a ROMmed system.

FLOATS ANS

$n1 - n2$

$n2$ is the size in address units of $n1$ floating-point numbers.

FLOG ANS

$r1 - r2$

$r2$ is the base-ten logarithm of $r1$.

FLOOR ANS

$r1 - r2$

Round $r1$ to an integral value using the “round toward negative infinity” rule, giving $r2$.

FMAX ANS

$r1 \ r2 - r3$

$r3$ is the greater of $r1$ and $r2$.

FMIN ANS

$r1 \ r2 - r3$

$r3$ is the lesser of $r1$ and $r2$.

Supplement 2

FNEGATE ANS

$r1 - r2$

$r2$ is the negation of $r1$.

FOVER ANS

$r1\ r2 - r1\ r2\ r1$

Place a copy of $r1$ on top of the floating-point stack.

FP FP2 FP3 FP4 FP5 FP6 FP7

$r -$

These are **MARKER** words to cut back the compiled floating point package. **FP** will remove all of it including subsequent definitions. **FP3** will leave the files **#FP1.TDS** and **#FP2.TDS** etc.

FROT ANS

$r1\ r2\ r3 - r2\ r3\ r1$

Rotate the top three floating-point stack entries.

FROUND ANS

$r1 - r2$

Round $r1$ to an integral value using the “round to nearest” rule, giving $r2$.

FS. ANS

r –

Display, with a trailing space, the top number on the floating-point stack in scientific notation:

<significand><exponent>

where:

<significand> := [-]<digit>.<digits0>

<exponent> := E[-]<digits>

See: REPRESENT .

FSIN ANS

r1 – r2

r2 is the sine of the radian angle r1.

FSINCOS ANS

r1 – r2 r3

r2 is the sine of the radian angle r1. r3 is the cosine of the radian angle r1.

See: FATAN2 .

FSINH ANS

r1 – r2

r2 is the hyperbolic sine of r1.

FSQRT ANS

r1 – r2

r2 is the square root of r1.

Supplement 2

FSWAP ANS

r1 r2 – r2 r1

Exchange the top two floating-point stack items.

FTAN ANS

r1 – r2

r2 is the tangent of the radian angle r1.

FTANH ANS

r1 – r2

r2 is the hyperbolic tangent of r1.

FVARIABLE ANS

“<spaces>name” –

Create a definition for name with the execution semantics defined below. Reserve 6 bytes of data space. name is referred to as an “f-variable.”

name Execution: – aa

aa is the address of the data space reserved by FVARIABLE when it created name. A program is responsible for initialising the contents of the reserved space.

Typical use: FVARIABLE name

F~ ANS

r1 r2 r3 – flag

This provides the three types of “floating point equality” in common use: “close” in absolute terms, exact equality as represented, and “relatively close”.

If r3 is positive, flag is true if the absolute value of (r1 minus r2) is less than r3.

If r3 is zero, flag is true if the encoding of r1 and r2 are exactly identical.

If r3 is negative, flag is true if the absolute value of (r1 minus r2) is less than the absolute value of r3 times the sum of the absolute values of r1 and r2.

PI PI/2 PI/4

– n

Useful floating point constants. `PI` is 3.1415926 etc.

PRECISION ANS

– u

Return the number of significant digits currently used by `F.` , `FE.` , or `FS.` as u.

RAD>DEG

r1 – r2

r2 is r1 radians converted to degrees.

Supplement 2

REPRESENT ANS

r ca u – n flag1 flag2

Provides a primitive for floating-point display. At ca, place the character-string external representation of the significand of the floating-point number r. Return the decimal-base exponent as n, the sign as flag1 and “valid result” as flag2. On completion the character string will contain the u most significant digits of the significand represented as a decimal fraction with the implied decimal point to the left of the first digit, and the first digit zero only if all digits are zero. The significand is rounded to u digits following the “round to nearest” rule; n is adjusted, if necessary, to correspond to the rounded magnitude of the significand. If flag2 is true then r was in the implementation-defined range of floating-point numbers. If flag1 is true then r is negative.

S>F

n – r

r is the floating point equivalent of 16-bit integer n.

SET-PRECISION ANS

u –

Set the number of significant digits currently used by F. , FE. , or FS. to u.

SF! ANS

r aa –

Store the floating-point number r as a 32-bit IEEE single-precision number at aa. Since the significand of the internal representation of r has more precision than the IEEE single-precision format, it is rounded using the “round to nearest” rule.

SF@ ANS

aa – r

Fetch the 32-bit IEEE single-precision number stored at aa to the floating-point stack as r in the internal representation.

SFALIGN ANS

–

If the data-space pointer is not single-float aligned, reserve enough data space to make it so.

SFALIGNED ANS

ca – aa

aa is the first single-float-aligned address greater than or equal to ca.

SFLOAT+ ANS

aa1 – aa2

Add the size in address units of a 32-bit IEEE single-precision number to aa1, giving aa2.

SFLOATS ANS

n1 – n2

n2 is the size in address units of n1 32-bit IEEE single-precision numbers.