PROPOSAL FOR AN OPTIONAL IEEE 754, BINARY FLOATING-POINT WORD SET

version **0.5.5**
31-Aug-09

drafted by David N. Williams, with several `comp.lang.forth` contributors, especially Andrew Haley, Anton Ertl, Ed, and Marcel Hendrix (alphabetical order)

TABLE of CONTENTS

1 INTRODUCTION

This is a proposal for an optional Forth 200x word set, called the "IEEE Floating-Point word set", that supports the binary part of the IEEE 754-2008 standard for floating-point arithmetic [1]. The most recent, freely available, but less comprehensive version is IEEE 754 draft 1.2.9, January 27, 2007 [2]. There is also a Wikipedia summary [3].

The standard [1] is hereafter referred to as "IEEE 754-2008", with section numbers indicated by IEEE 754-2008 <number>.

This specification requires that ISO Forth [4,5] floating-point and floating-point extension words in the optional Floating-Point word set, when present with the IEEE Floating-Point word set, satisfy additional IEEE 754-2008 requirements. Words in that word set and this that correspond to mathematical, including logical, operations or functions in IEEE 754-2008 adopt the behavior required or recommended there by reference, unless otherwise stated.

The specification is compatible with, rather than conformant to, IEEE 754-2008, because it includes only a subset of the IEEE requirements. It also aims to make many of the remaining requirements expressible in Forth.

Reference [4], the final draft of "ANSI X3.215-1994, American National Standard for Information Systems—Programming Languages--Forth", is hereafter referred to as "DPANS94". It is believed to be the same as the published version, ISO/IEC 15145:1997 [5]. This document adopts the official terminology of DPANS94 unless otherwise stated. Section numbers in that document are indicated by DPANS94 <number>.

The meaning of "optional" for these word sets is defined by following two paragraphs from DPANS94 A.1.3.1:

The basic requirement is that if the implementor claims to have a particular optional word set the entire required portion of that word set must be available. If the implementor wishes to offer only part of an optional word set, it is acceptable to say, for example, "This system offers portions of the [named] word set", particularly if the selected or excluded words are itemized clearly.

and

Optional word sets may be offered in source form or otherwise factored so that the user may selectively load them.

The current C99 standard [6-8], ISO/IEC 9899:1999, has a comprehensive treatment of IEEE 754-1985, which offers a route to implementation for those Forth systems that can call C libraries. Reference [7] is believed to faithfully reflect the current C99 standard. Section numbers from reference [7] are indicated by C99 WG14/N1256 <number>.

[** Bracketed statements like this are for editorial questions and comments, eventually to be removed.**]**

2 TERMINOLOGY AND NOTATION

fp: Short for "floating point". The Forth floating-point stack is called the "fp stack". In this document, synonymous with "binary floating point".

IEEE special datum, or an **IEEE special**: Signed zero, a quiet or signaling signed nan, or signed infinity.

full IEEE set: For an IEEE binary format, the set of normal and subnormal numbers plus special data that it represents.

IEEE datum: Any member of a full IEEE set.

IEEE arithmetic: Arithmetic defined by IEEE 754-2008 for IEEE data.

affinely extended reals: Finite real numbers and +/-infinity, with
-infinity < {every finite number} < +infinity.

nan load or **nan payload**: The value of the fractional bits in the binary format of a nan, excluding the quiet bit, considered as a positive integer. The smallest signaling load is unity, and the smallest quiet load is zero.

qnan, **snan**: A quiet or signaling nan, respectively, of any sign or load.

single: In the context of Forth fp, an IEEE 754-2008 32-bit interchange format.

double: In the context of Forth fp, an IEEE 754-2008 64-bit interchange format.

default: In the context of Forth fp, the float format for data that can appear on the fp stack.

fp exception: Unless otherwise stated, never used in this document in the sense of Forth `CATCH` and `THROW`, but always in the sense of IEEE 754-2008 2.1.18 exception: An event that occurs when an operation on some particular operands has no outcome suitable for every reasonable application. That operation might signal one or more exceptions by invoking the default or, if explicitly requested, a language-defined alternate handling. Note that "event", "exception", and "signal" are defined in diverse ways in different programming environments.

fp status flag: C99: WG14/N1256 7.6: A floating-point status flag is a system variable whose value is set (but never cleared) when a floating-point exception is raised, which occurs as a side effect of exceptional floating-point arithmetic to provide auxiliary information.

fp control mode: C99: WG14/N1256 7.6: A floating-point control mode is a system variable whose value may be set by the user to affect the subsequent behavior of floating-point arithmetic. Only rounding control modes are addressed in this document. In particular, alternate exception handling modes are not included.

correct rounding: Conversion of an infinitely precise result to a floating-point number or infinity according to the current rounding mode.

3 BINARY FLOATING-POINT FORMATS

Each IEEE binary fp format has two fixed parameters, $p > 0$ (precision) and $e_{max} > 0$ (maximum exponent), and defines $e_{min} = 1 - e_{max}$ (minimum exponent). Each such format represents all real numbers of the form

$$r = (-1)^{s} * 2^{e} * b_0.b_1 \ldots b_{p-1}$$

where

$$s = 0 \text{ or } 1, \quad e_{min} \le e \le e_{max}, \quad b_i = 0 \text{ or } 1, \quad p = \#\text{ significand bits.}$$

See Section A.3 for more information about IEEE binary fp formats.

The binary fp formats in this document are to be regarded as logical formats defined only by p and e_{max}, with unspecified encoding or layout in memory or on the fp stack. In particular, neither the presence or absence of an explicit integer bit, b_0, nor endianness, is specified.

Numbers with $b_0 = 1$ are called “normal". Numbers with $b_0 = 0$ and $e = e_{min}$ are called "subnormal".

4 IMPLEMENTATION

4.1 Requirements

The DPANS94 floating-point and floating-point extensions word sets are optional word sets, and so are the word sets described by this document.

The word "shall" in the remainder of this document states a requirement when the environmental query for IEEE-FP or IEEE-FP-EXT returns true. "Should" means "strongly recommended".

4.2 Default format

Default fp data, i.e., data that can appear on the fp stack, shall correspond to one of the IEEE basic or extended, full logical binary formats.

The fp stack shall be separate from the data stack, in accord with the proposal accepted at the 2008 Forth200x meeting.

Data stored in memory by `F!` shall have the default logical format.

4.3 Accuracy

Unless otherwise stated, the accuracy of floating-point calculation is left to the implementation. This is inevitable because of practical limitations on the computational state of the art.

4.4 Rounding

IEEE 754-2008 specifies the following settable rounding modes:

```
roundTiesToEven
roundTowardPositive
roundTowardNegative
roundTowardZero
roundTiesToAway
```

The last of these seems to be new, and is not addressed in this document. The first four are commonly implemented in hardware that claims IEEE compliance.

IEEE 754-2008 requires `roundTiesToEven` as the default mode for binary rounding, and recommends that for decimal rounding. This document adopts the same. `roundTiesToEven` shall be the default for binary rounding and should be the default for

decimal rounding.

In DPANS94, a few words explicitly use "round to nearest", in the same sense as `roundTiesToEven`; and a few words have explicit requirements for integer rounding. Otherwise, rounding is specified as implementation defined.

[** DPANS94 12.3.1.2:
… Any rounding or truncation of floating-point numbers is implementation defined.
]

Many IEEE floating-point operations specify that the current rounding mode shall be used. This document adopts that whenever relevant.

Using the current rounding mode affects text interpretation, arithmetic operations, math functions except nearest integer functions, and the following words, listed according to DPANS94 rounding:

`REPRESENT  DF!  DF@  SF!  SF@`	round to nearest
`>FLOAT  F.  FE.  FS.`	Implementation defined

In this specification, the words in the round to nearest list above remain compatible with DPANS94 programs when the current rounding mode is the default mode. The rounding mode can be changed from the default mode only by words in the optional IEEE Floating-Point extensions word set, none of which is present in DPANS94.

Ideally, mathematical functions ought to be "correctly rounded", i.e., computed with infinite accuracy and then rounded in the current mode. Where accuracy is implementation defined, that may not be strictly achievable.

4.5 IEEE Exceptions

IEEE floating-point exceptions have two aspects, signaling and flags.

Signaling simply means handling the exception.

There are flags for five exceptions, all commonly implemented in hardware that claims IEEE compliance:

`invalid`,
`divideByZero`,
`overflow`,
`underflow`, and
`inexact`.

Flags can be raised by an IEEE compliant system or by a user program. The act of raising a flag does not cause any other action. Reading a flag does not change it. A flag can be lowered only by explicit user request.

IEEE specifies the fp operations and conditions that signal exceptions. According to IEEE 754-2008 7.1, "Overview: exceptions and flags", the default handling is nearly always to produce a default datum, raise the corresponding flag, and keep going.

IEEE recommends that a language standard should define alternate, nondefault handling for fp exceptions, with guidelines that allow stopping program execution and reporting an error **[**** IIUC **]**.

This document specifies the default to be IEEE default handling, invoked under the conditions required by IEEE 754-2008 unless otherwise stated, with the aim of restricting alternate handling as little as possible. An interface for alternate handling **[**** not yet specified**]** should interact with the system at a low level to change the default handling of exceptions in basic fp operations such as `F+` and `F*`. For situations where there would be little performance penalty, like text interpretation, the interface might cater to `CATCH` and `THROW`.

The exception flags can be accessed only by the optional IEEE Floating-Point extensions word set. When those words are not present, the IEEE requirements for setting the flags are moot, because the flags have no side effects. Then, except for a few operations that continue to specify DPANS94 ambiguous conditions, the only noticeable effect of exceptions is to produce IEEE special data.

5 DATA TYPES

For the purpose of this document, the DPANS94 `r` type is extended to include all IEEE data for the default fp format.

6 ENVIRONMENTAL QUERIES

If either of the first two queries returns `true` for the "known" flag, then so shall the other. If the third query returns true, then so shall the other two.

String	Value Data Type	Constant?	Meaning
IEEE-FP	flag	no	IEEE and DPANS94 Floating-Point wordsets present
IEEE-FP-FORMAT	d	no	in usual stack notation, the default format has IEEE parameters (emax p)
IEEE-FP-EXT	flag	no	IEEE Floating-Point extensions word set present

A true result for the `IEEE-FP` environmental query shall mean that any words that are present

from the DPANS94 Floating-Point word set, the DPANS94 Floating-Point extensions word set, or the IEEE Floating-Point word set obey the specifications of this document, and that generic specifications not related to the presence or absence of particular words are satisfied.

It shall also mean that the DPANS94 `MAX-FLOAT` query shall return true and the largest, finite number in the default format.

The data value for the `IEEE-FP` query is true if and only if all words in the DPANS94 and IEEE Floating-Point word sets are present.

Nothing in this document depends on the encoding of the logical format corresponding to the values of $e_{\max}$ and p returned as data by the `IEEE-FP-FORMAT` query.

The data value for the `IEEE-FP-EXT` query is true if and only if all words in IEEE Floating-Point extensions word set are present.

7 TEXT INPUT

7.1 Constants

```
+INF          ( f: -- +Inf )
-INF          ( f: -- -Inf )
+NAN          ( f: -- +NaN )
-NAN          ( f: -- -NaN )
```

These words return, respectively, IEEE signed infinity and the quiet signed nan with zero load, in the default format.

See Section A.7.1 for more information about the encoding of NaN.

7.2 Decimal input

IEEE requires that conversion between text and binary fp formats shall include signed zero, signed infinity, and signed nans, with and without loads. See IEEE 754-2008 5.4.2, "Conversion operations for floating-point formats and decimal character sequences", and 5.12, "Details of conversion between floating-point data and external character sequences".

Conversion of nan loads is not included in this specification. Signed infinity and signed, quiet, unloaded nans are covered by the constants defined in Section 7.1. Signed zero is already included in the syntax specification in DPANS94 12.3.7, "Text input number conversion".

DPANS94 12.3.7 specifies that the number-conversion algorithm used by the text interpreter is to be extended to recognize floating-point numbers when the base is decimal, with an implication that the behavior on failure is to be governed by the introductory paragraphs in DPANS94 3.4, "The Forth interpreter". This document slightly modifies the syntax specification, and is more explicit about failure.

When IEEE-FP is present, the syntax specification in DPANS94 12.3.7 shall be replaced by:

Convertible string := <significand><exponent>

```
<significand> := [<sign>]<digits>[.<digits0>]
<exponent>    := <e-char>[<sign>]<digits0>
<digits>      := <digit><digits0>
<digits0>     := <digit>*
<sign>        := { + | - }
<e-char>      := { E | e }
<digit>       := { 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 }
```

The only change in syntax is the additional "e" option in <e-char>.

Interpretation shall convert to an affinely extended real in the default floating-point format, and

shall be correctly rounded. Inexact, overflow, and underflow conditions shall be signaled in the IEEE sense.
[** ALTERNATIVE: (still under discussion in `comp.lang.forth`) no IEEE signaling, and an ambiguous condition on overflow.**]**

An ambiguous condition exists if the syntax is not satisfied and causes unsuccessful text intepretation, in which case behavior shall be governed by DPANS94 3.4.4, "Possible actions on an ambiguous condition".

[** QUOTE
3.4.4 Possible actions on an ambiguous condition

When an ambiguous condition exists, a system may take one or more of the following actions:

- ignore and continue,
- display a message,
- execute a particular word,
- set interpretation state and begin text interpretation,
- take other implementation-defined actions,
- take implementation-dependent actions.

The response to a particular ambiguous condition need not be the same under all circumstances.
]

7.3 Hexadecimal input

IEEE requires a text format for numbers with a hexadecimal significand, and decimal radix two exponent, with exact conversion to and from binary fp formats where possible. See IEEE 5.12.3, "External hexadecimal-significand character sequences representing finite numbers".

Conversion of that format is not included in this specification.

8 IEEE FLOATING-POINT WORDS

This is an optional word set.

Unless otherwise stated, all fp words that do computations or comparisons shall obey the requirements and recommendations of IEEE 754-2008 5 and 6, for binary formats.

8.1 Conversion

`D>F` `( d -- ) ( f: -- r )`

The result `r` is the floating-point equivalent of `d`. If conversion of `d` cannot be represented precisely in the default fp format, it shall be rounded according to the current rounding mode, and the inexact or overflow exception shall be handled just as for arithmetic operations. See IEEE 754-2008 5.4.1, "Arithmetic operations",

`formatOf-convertFromInt(int).`

[** RATIONALE: Default handling for `overflow/inexact` is given by IEEE 754-2008 7.4/7.6. The fp status flags are set, and overflow produces +/-infinity, while `inexact` rounds. Default overflow handling sets the `inexact` flag as well.

Note that `d` is always in range for any of the formats binary32, binary64, binary80, and binary128, for 32- or 64-bit systems.

A programmer can assume that

```
( d) 2DUP D>F F>D D= ( flag)
```

results in a true flag *only* when $| d | < 2^{(p+1)}$, where p is the format precision, so that rounding does not occur. Rounding does occur, for example, with 32-bit systems and binary64, when $| d | >= 2^{54}$.
]

`F>D` `( f: r -- ) ( -- d )`

The result `d` is the double-cell signed-integer equivalent of the integer portion of `r`. The fractional portion of `r` is discarded. An ambiguous condition exists if the integer portion of `r` cannot be precisely represented as a double-cell signed integer, or if `r` is a `nan` or `infinity`. **[****The only noncosmetic change is the last clause of the last sentence.**]**

[** Marcel Hendrix suggests the following reasonable behavior for the ambiguous condition on two's complement systems:

When the integer part of r is not representable by a signed, double number, or when it is a NaN or Inf, the following happens:

```
FORTH> +Inf F>D hex UD. 8000000000000000  ok
FORTH> -Inf F>D hex UD. 8000000000000000  ok
FORTH> +NaN F>D hex UD. 8000000000000000  ok
FORTH> -NaN F>D hex UD. 8000000000000000  ok
```

]

[** RATIONALE: The only change from DPANS94 is to include nans and infinity in the ambiguous condition. Except for exception handling, F>D corresponds to the IEEE function convertToIntegerTowardZero().

IEEE requires that the invalid operation exception be signaled when r is a nan or infinity or out of range of the destination format. See IEEE 754-2008 5.8, "Details of conversions from floating-point to integer formats".

IEEE also requires

```
convertToIntegerTiesToEven()
convertToIntegerTowardPositive()
convertToIntegerTowardNegative()
convertToIntegerTiesToAway()
```

plus versions of the five conversions that signal inexact when appropriate. Note that, except for convertToIntegerTiesAway and the Forth ambiguous condition, the above conversions are equivalent to Forth phrases such as "FCEIL F>D".

]

```
>FLOAT          ( c-addr u -- true|false ) ( f: -- r| )
```

The phrase "the string represents a valid floating-point number" in DPANS94 12.6.1.0558 shall be interpreted to mean that it represents a finite number in the range of the default format.

```
>IEEEFLOAT      ( c-addr u -- true|false ) ( f: -- r| )
```

This word extends the functionality of >FLOAT to include IEEE special data, with modified syntax.

The decimal to binary conversion shall use the current rounding mode.

Overflow, underflow, and inexact exceptions shall be treated according to IEEE 754-2008 7.4, "Overflow", 7.5 "Underflow", and 7.6 "Inexact".

If the string represents a real number in the syntax below, the result of conversion to the default format (signed infinity on overflow) and true are returned.

If the string represents an IEEE special datum in the syntax below, the datum and true are returned, with no change in the fp exception status. A string of blanks **[**** or the empty string?**]** shall be treated as +0.

Syntax of a convertible string `:= { <significand>[exponent] | <special> }`

```
  <significand> := [<sign>]{ <digits>[.<digits0>] | .<digits> }
  <exponent>    := <marker><digits0>
  <marker>      := { <e-form> | <sign> }
  <e-form>      := <e-char>[<sign>]
  <sign>        := { + | - }
  <e-char>      := { D | d | E | e }
  <special>     := [<sign>]{ <inf> | <nan> }
  <inf>         := { Inf | inf | INF | infinity | Infinity }
  <nan>         := { NaN | nan | NAN }

REPRESENT       ( f: r -- )  ( c-addr u -- n flag1 flag2 )
```

When `flag2` is true, the floating-point datum `r` was a finite number; when `false`, it was a nan or infinity. In any case, `flag1` is true if and only if the sign bit of `r` was unity.

When `r` is a finite number, place the character-string representation of its significand at `c-addr`, and return the decimal-base exponent as `n`. The character string shall consist of 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 shall be correctly rounded to `u` digits, with correct signaling for inexact and underflow conditions; `n` is adjusted, if necessary, to correspond to the rounded magnitude of the significand.

If `u` is greater than or equal to

$$M = 1 + \text{ceiling}(p * \log_{10} (2)) ,$$

where p is the precision returned by the `IEEE-FP-FORMAT` environmental query, and the rounding mode is `roundTiesToEven` in both directions, conversion back to binary with the algorithm of `>FLOAT` or the text interpreter shall reproduce `r`. The string shall be padded with $u - M$ trailing zeroes if u is larger than M.

An ambiguous condition exists if the value of `BASE` is not decimal ten.

When `r` is a nan or infinity, the character string shall be one of the three-character options for `<nan>` or `<inf>`, respectively, recognized by `>FLOAT`. If `u` is less than three, the string is truncated; if more than three, it is padded with trailing blanks. When `r` is infinity, the value `n` returned is zero; when `r` is a nan, it is a nonzero, implementation-defined number. In neither case is an exception signaled.

[** There is ongoing discussion about this in `comp.lang.forth`.**]**

[** RATIONALE: The implementation-defined value of n for nans could be used in a future proposal to encode information about the signaling bit and load.

Correct rounding with IEEE floating-point exceptions is required by IEEE-754-2008 5.12.2, "External decimal character sequences representing finite numbers". Among the exceptions specified in IEEE-754-2008 4.3, "Rounding-direction attributes", overflow is irrelevant for finite `r`, leaving inexact and underflow. Infinity and nan are treated explicitly.

Reproducibility under binary to decimal to binary conversion is required by IEEE 754-2008 5.12.2. It ensures that a user can print values and read them back without loss of precision.

A reference implementation which demonstrates correct rounding and reproducibility remains to be presented. The following example uses `REPRESENT` to define a word that prints decimal floating-point output with the maximum usable number of digits.

```
s" IEEE-FP-FORMAT" ENVIRONMENT? [IF] ( emax p) NIP
0 D>F 2E FLOG F* FCEIL 1E F+ FCONSTANT MAX-FLOAT-DIGITS [THEN]

: .SIGN  ( flag -- )  IF  ." -"  ELSE  ." +"  THEN ;

: F.BIG  ( r -- )
  PAD MAX-FLOAT-DIGITS REPRESENT IF
      .SIGN ." ."
      PAD MAX-FLOAT-DIGITS TYPE
      ." E" 0 .R
    ELSE
      OVER 0= ( INF) IF  DUP .SIGN  THEN
      2DROP  PAD MAX-FLOAT-DIGITS TYPE
    THEN ;
]

SF!             ( f: r -- )    ( sf-addr -- )
SF@             ( sf-addr -- ) ( f: -- f )
DF!             ( f: r -- )    ( df-addr -- )
DF@             ( df-addr -- ) ( f: -- f )
```

The specification for these DPANS94 words is amended to explicitly require conversion to or from the respective IEEE 754-2008 binary32 or binary64 logical interchange formats, with implementation-defined memory layout. The conversion shall be exact in either direction for signed zero, signed infinity, and real numbers to a wider format, and shall use the current rounding mode for conversion of real numbers to a narrower format (see IEEE 754-2008 5.4.2, formatOf-convertFormat). The conversion of nans is implementation defined, but should not signal an exception, should preserve the sign bit, and should treat payloads according to IEEE 754-208 6.2.3, "NaN propagation".

8.2 Output

```
F.              ( f: r -- )
FE.             ( f: r -- )
FS.             ( f: r -- )
```

The DPANS94 specification is extended to include IEEE specials, with output text of the appropriate form below, with implementation-dependent case sensitivity:

```
[<sign>]0{ E | e }<space>
[<sign>]{ Inf | INF | inf }<space>
[<sign>]{ NaN | NAN | nan }<space>
```

The current rounding mode shall be used.

[** RATIONALE: The use of the current rounding mode is a change from DPANS94, where rounding is implementation defined.**]**

8.3 Comparison

IEEE has twenty-two required comparisons which apply to the full set of IEEE data. Twelve of these are quiet, and ten are signaling. See IEEE 754-2008 5.6.1, "Comparisons", and 5.11, "Details of comparison predicates".

This proposal requires only quiet comparisons, which do not signal exceptions, and of those, only a subset of five, which is sufficient for expressing all twelve.

See Section A.8 for rationale and more information about the remaining comparisons, with high-level implementation examples.

IEEE identifies four fundamental, mutually exclusive comparisons: less than ("<"), equal ("="), greater than (">"), and unordered (see rule 3 below). Each of these is true iff each of the others is false.

The basic rules are the following:

1. The sign of zero is ignored.
2. The sign of infinity is not ignored, and is treated in the natural way for the "ordinary" comparisons with real numbers or infinity, namely <, =, and >. In particular, either signed infinity is equal to itself.
3. The unordered comparison is true iff at least one of its two arguments is a nan. That implies that any of the other three, "ordinary" comparisons involving a nan is false.

The five required comparisons are "<", ">", "=", "<=", and ">=", where "<=" and ">=" stand for the usual phrases "less than or equal" and "greater than or equal". Note that familiar identities for real numbers are generally not satisfied by IEEE comparisons. For example, the negation of "<" is not the same as ">=". See Section A.8.

```
F<      ( f: r1 r2 -- ) ( -- [r1<r2]? )
F=      ( f: r1 r2 -- ) ( -- [r1=r2]? )
F>      ( f: r1 r2 -- ) ( -- [r1>r2]? )
F<=     ( f: r1 r2 -- ) ( -- [r1<=r2]? )
F>=     ( f: r1 r2 -- ) ( -- [r1>=r2]? )
F0<     ( f: r -- ) ( -- [r<0]? )
F0=     ( f: r -- ) ( -- [r=0]? )
F0>     ( f: r -- ) ( -- [r>0]? )
F0<=    ( f: r -- ) ( -- [r<=0]? )
F0>=    ( f: r -- ) ( -- [r>=0]? )
```

The data stack outputs are DPANS94 flags corresponding to the indicated IEEE predicates. In particular, the specifications for the existing DPANS94 words `F<`, `F0<`, and `F0=` are extended to include IEEE specials.

```
F~    ( f: r1 r2 r3 -- ) ( -- flag )
```

If `r3` has positive sign and is neither a `nan` nor zero, `flag` is true iff the absolute value of `r1` minus `r2` is less than `r3`, taking into account IEEE arithmetic and comparison rules.

If `r3` is signed zero, `flag` is true iff `r1` and `r2` have identical formats.

If `r3` has negative sign and is neither a `nan` nor zero, `flag` is true iff 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`, taking into account IEEE arithmetic and comparison rules.

If `r3` is a nan, `flag` is false.

8.4 Classification

IEEE 754-2008 5.7.2, "General operations", requires a large number of classification operations. This documents defines only those corresponding to:

```
isSignMinus
isNormal
isFinite
isZero
isSubnormal
isInfinite
isNaN
```

Actually `isSignMinus` corresponds to `FSIGNBIT`, and `isZero` corresponds to `F0=`, which leaves the following:

```
FINITE?         ( f: r  -- ) ( -- [normal|subnormal]? )
FNORMAL?        ( f: r  -- ) ( -- normal? )
FSUBNORMAL?     ( f: r  -- ) ( -- subnormal? )
FINFINITE?      ( f: r  -- ) ( -- [+|-]Inf? )
FNAN?           ( f: r  -- ) ( -- nan? )
```

8.5 Arithmetic

See IEEE 5.4.1, "Arithmetic operations".

```
F*              ( f: r1 r2 -- r1*r2 )
F*+             ( f: r1 r2 r3 -- [r2*r3]+r1 )
F+              ( f: r1 r2 -- r1+r2 )
F-              ( f: r1 r2 -- r1-r2 )
F/              ( f: r1 r2 -- r1/r2 )
FSQRT           ( f: r -- sqrt[r] )
```

The DPANS94 specification is extended to IEEE arithmetic. These operations shall be correctly rounded. See IEEE 754-2008 5.1, "Overview", for precision, rounding, special data treatment, and exceptions and 5.4.1, "Arithmetic operations", for the arithmetic words.

8.6 Math functions

The Forth words FABS, FMAX, FMIN, and FSQRT are covered elsewhere.

The DPANS94 specification for the following words is extended to adopt the corresponding IEEE behavior. These words should be correctly rounded, and shall use current rounding. See IEEE 754-2008 9.2, "Recommended correctly rounded functions", and 9.2.1, "Special values".

```
F**  FACOS  FACOSH  FALOG  FASIN  FASINH  FATAN  FATAN2
FATANH  FCOS  FCOSH  FEXP  FEXPM1  FLN  FLNP1  FLOG  FSIN
FSINCOS  FSINH  FSQRT  FTAN  FTANH
```

8.7 Sign bit operations

```
FSIGNBIT        ( f: r  -- ) ( -- minus? )
```

This word corresponds to `isSignMinus` in IEEE 754-2008 5.7.2, "General operations". The name is based on C99.

The following are all required by IEEE. See IEEE 5.5.1, "Sign bit operations". The IEEE `copy()` function is superfluous in Forth **[**** IIUC **]**.

```
FNEGATE         ( f: r -- -r )
FABS            ( f: r -- |r| )
```

The DPANS94 specification is extended to IEEE specials.

```
FCOPYSIGN       ( f: r1 r2 -- r3 )
```

The output `r3` is `r1` with its sign bit replaced by that of `r2`.

8.8 Nearest integer functions

All of the words in this section correspond to C99 functions with similar names, including the already existing `FLOOR` and `FROUND` from DPANS94 and `FTRUNC` from Forth 200x, except that the C99 `round()` does `roundToIntegralTiesToAway` instead of `roundToIntegralTiesToEven`.

```
FCEIL           ( f: r1 -- r2 )
FLOOR           ( f: r1 -- r2 )
FROUND          ( f: r1 -- r2 )
FTRUNC          ( f: r1 -- r2 )
```

These words correspond to the respective IEEE required operations:

```
    roundToIntegralTowardPositive
    roundToIntegralTowardNegative
    roundToIntegralTiesToEven
    roundToIntegralTowardZero
```

See IEEE 754-2008 5.3.1, "General operations" and 5.9, "Details of operations to round a floating-point datum to integral value". No word is defined for IEEE `roundToIntegralTiesToAway`.

```
FNEARBYINT      ( f: r1 -- r2 )
```

This word corresponds to the IEEE required operation:

```
    roundToIntegralExact
```

It performs the function of whichever of the other four corresponds to the current rounding mode.

8.9 Data manipulation

```
FMAX            ( f: r1 r2 -- r3 )
FMIN            ( f: r1 r2 -- r3 )
```

The DPANS94 specification is extended to IEEE specials. See `minNum` and `maxNum` in IEEE 754-2008 5.3.1, "General operations" and 6.2, "Operations with NaNs".

FNEXTUP `( f: r1 -- r2 )`
FNEXTDOWN `( f: r1 -- r2 )`

When `r1` is a nonzero real number, `FNEXTUP` returns the next affinely extended real in the default format that compares larger than `r1`, and `FNEXTDOWN` returns the next one that compares less than `r1`. See IEEE 754-2008 5.3.1, "General operations" for the behavior when `r1` is an IEEE special.

[** RATIONALE: `FNEXTDOWN` can be defined as:

```
: FNEXTDOWN  ( f: r1 -- f2 )  FNEGATE FNEXTUP FNEGATE ;
```

For accuracy control, it is useful to have efficient implementations of both words.
]

FSCALBN `( f: r -- r*2^n ) ( n -- )`

The output is efficiently scaled by 2^n. See IEEE 754-2008 5.3.3,"logBFormat operations".

FLOGB `( f: r -- e )`

Leave the radix-two exponent `e` of the fp representation as an fp integer. If `r` is subnormal, the exponent is computed as if `r` were normalized, with `e < emin`. See IEEE 754-2008 5.3.3, "logBFormat operations" for treatment of IEEE specials.

FREMAINDER `( f: x y -- r q )`

When `y` is not 0, the remainder `r = fremainder(x, y)` is defined for finite `x` and `y` regardless of the current rounding mode by the exact mathematical relation $r = x - y * q$, where q is the integer nearest the exact number x/y, with `roundToIntegralTiesToEven`. If $r = 0$, its sign shall be that of x, and `fremainder(x, inf)` is x for finite x. See IEEE 754-2008 5.3.1, "General operations".

9 IEEE FLOATING-POINT EXTENSION WORDS

This is an optional word set.

9.1 Status flags

A floating-point status flag is raised, but never lowered, as a side effect of an fp operation. The default side effect also provides a default fp datum and does not interrupt program flow.

Floating-point status flags can be raised or lowered by `SET-FSTATUS`, and read by `GET-FSTATUS`. Both words have normal interpretation and compilation semantics.

In the following, "fmask" stands for "floating-point status mask", and "fflags" stands for the bitwise OR of flag data corresponding to an fmask.

The fp status flags correspond to the following distinct, nonzero fp flag masks:

`FDIVBYZERO FINEXACT FINVALID FOVERFLOW FUNDERFLOW`

Masks may be OR'd to form fmask's.

[** RATIONALE: The mask names correspond to the C99 macros

`FE_DIVBYZERO`, `FE_INEXACT`, `FE_INVALID`, `FE_OVERFLOW`, and `FE_UNDERFLOW`,

based on C99:WG14/N1256 7.6.2.

The masks have nothing to do with the corresponding cpu masks in intel processors, which select the default or an alternate IEEE fp exception handler.
]

`GET-FSTATUS` `( fmask -- fflags )`

Return the bit-wise OR of the current fp status flags selected by the nonzero bits of `fmask`, without changing the status. It is an ambiguous condition if `fmask` is not the OR of a subset of the five, named masks, or if the underlying system does not support all five flags.

`SET-FSTATUS` `( fflags fmask -- )`

Set the values of the fp status flags selected by `fmask` to the values of the corresponding bits in `fflags`, without side effects. It is an ambiguous condition if `fmask` is not the OR of a subset of the five, named masks, or if the underlying system does not support all five flags.

[** MORE RATIONALE: The spirit of IEEE 754-2008 exceptions is that processing continues without interrupting program flow. A default fp datum is returned, and an exception flag is set, along with other possible status information. Applications can query such results and react to

them. This scheme provides great flexibility.

IEEE 754-2008 also allows the implementation of nondefault exception handlers, which may interrupt program flow. C99 has provisions for switching to such a mode, but does not require it, and seems vague about how it's to be done.

The proposed words implement the simplest model, where the only information provided about the fp exception state is the flag values. The C99 functions `fegetexceptflag()` and `fesetexceptflag()` are models for `GET-FPSTATUS` and `SET-FPSTATUS`. Extra words that allow more information could be proposed in the future, if needed.

The following quote from the C99 standard describes the intended programming style, both for fp status flags and rounding modes.

C99 WG14/N1256 7.6, "Floating-point environment <`fenv.h`>", paragraph 2:

Certain programming conventions support the intended model of use for the floating-point environment:

- a function call does not alter its caller's floating-point control modes, clear its caller's floating-point status flags, nor depend on the state of its caller's floating-point status flags unless the function is so documented;

- a function call is assumed to require default floating-point control modes, unless its documentation promises otherwise;

- a function call is assumed to have the potential for raising floating-point exceptions, unless its documentation promises otherwise.

With these conventions, a programmer can safely assume default floating-point control modes (or be unaware of them). The responsibilities associated with accessing the floating-point environment fall on the programmer or program that does so explicitly.
]

9.2 Rounding modes

[** UNDER CONSTRUCTION This section is currently under discussion in `comp.lang.forth`.

See IEEE 9.3, "Operations on dynamic modes for attributes". Only words corresponding to 9.3.1, "Operations on individual dynamic modes", are expected to be implemented, and among those, `roundTiesToAway` is not expected to be implemented.
]

10 REFERENCES

1. "IEEE Standard for Floating-Point Arithmetic", approved June 12, 2008 as IEEE Std 754-2008: link to IEEE Std 754-2008

2. "DRAFT Standard for Floating-Point Arithmetic P754", IEEE 754 draft 1.2.9, January 27, 2007: link not valid

3. Wikipedia, "IEEE 754-2008": http://en.wikipedia.org/wiki/IEEE_754

4. ANSI X3.215-1994 final draft: http://www.taygeta.com/forth/dpans.html

5. ISO/IEC 15145:1997: link to ISO/IEC 15145:1997 (webstore.ansi.org), `link to ISO/IEC 15145:1997 (iso.org)`

6. ISO/IEC 9899:1999 (December 1, 1999),
   ISO/IEC 9899:1999 Cor. 1:2001(E),
   ISO/IEC 9899:1999 Cor. 2:2004(E),
   ISO/IEC 9899:1999 Cor. 3:2007(E): link to ISO/IEC 9899, 1999

7. C99, TC1, TC2, and TC3 are included in the freely available WG14/N1256, September 7, 2007 **[**** thanks to David Thompson**]**: link to C99 WG14/N1256

8. Single UNIX 3, AFAICS duplicates the C99 library spec, with some things pinned down more tightly **[**** thanks to David Thompson**]**: link not valid

9. "Intel(R) 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture", Table 4-3: http://www.intel.com/Assets/PDF/manual/253665.pdf

A.3 BINARY FLOATING-POINT FORMATS

IEEE 754-2008 defines three basic binary fp formats, binary32, binary64, and binary128, plus three corresponding extended binary formats, whose parameters are shown in Tables 1 and 2 below. It also defines the four binary interchange formats shown in Table 3, plus those with storage widths of more than 128 bits that are a multiple of 32 bits.

Table 1: Parameters for IEEE 754-2008 basic binary formats.

	binary32	binary64	binary128
p =	24	53	113
emax =	127	1023	16383

Table 2: Parameters for IEEE 754-2008 extended binary formats.

	binary32	binary64	binary128
p >=	32	64	128
emax >=	1023	16383	65535

Table 3: Parameters for IEEE 754-2008 binary interchange formats (k is the storage width in bits).

	binary16	binary32	binary64	binary128
k =	16	32	64	128
p =	11	24	53	113
emax =	15	127	1023	16383

Note that the intel 80-bit format corresponds to one of the extended binary64 formats, with $p = 64$ and $e_{\max} = 16383$. Its precision is greater than that of basic binary64 and less than that of basic binary128, with exponent range the same as basic binary128. Although it is not defined as a basic IEEE binary format, it may be called the "binary80" basic format. Its implementation normally differs from that of the other basic formats by having an explicit leading bit for normal and subnormal numbers.

Binary interchange formats are logical formats, with unspecified memory layout. They all have an implicit leading bit for normal and subnormal numbers.

Note that the binary128 interchange format is the only one in Table 3 that can contain the binary80 basic format. IEEE does not define a binary80 interchange format.

A.7.1 NAN SIGNS AND LOADS

IEEE allows the load for nan results like `0E 0E F/` to be anything, so the following does not necessarily give a zero load:

```
0E 0E F/ FABS FCONSTANT +NAN
```

Aside from the ambiguous load, the `FABS` (extended to nan) is necessary here, because not only does IEEE not specify the sign, both pfe and gforth actually give opposite signs for `0E 0E F/` under ppc (+) vs. intel (-) Mac OS X. They do both give quiet nans with zero load.

As a matter of fact, the intel QNaN "floating-point indefinite" is the qnan with zero load and negative sign [9].

A.8.3 COMPARISON

The twelve IEEE required comparisons are the following, where "N" means logical negation, "?" stands for "unordered", and "<?" and ">?" stand for "less than or unordered" and "greater than or unordered":

```
  <   =   >   ?   N<   N=   N>   N?   <=   >=   <?   >?
```

Unfortunately, the common notation "?" for the unordered predicate clashes with Forth practice, where "?" usually **[**** always?**]** indicates a flag or test. The ? notation is used here as a convenience for IEEE predicates, and does not appear in any corresponding Forth names.

The <= and >= comparisons are no longer simple negations of > and <, but are rather the AND's of those negations with N?. It can be shown that <? is N(>=), and >? is N(<=). See IEEE Table 5.3, "Required unordered-quiet predicates and negations".

It is possible to implement all of the IEEE comparisons via high-level definitions in terms of a few low-level words, even fewer than the five required for this word set. The following remarks are offered as a guide to possibilities for the choice of low-level words.

DPANS94 has only `F<` and `F~`. Since IEEE "=" is semantically different from `0E F~`, low-level implementation of `F=` seems inevitable. The two words `F<` and `F=` are probably a minimum set for high-level implementation of the rest.

For example, IEEE ">" is not expressible in terms of "<" and "=" plus logical operations, but

F> can be defined as:

```
: F>  ( f: r1 r2 -- ) ( -- [r1>r2]? )  FSWAP F< ;
```

FUNORDERED is not required by this document; in particular, the name is not reserved; but it can be defined as

```
: FUNORDERED  ( f: r1 r2 -- ) ( -- [r1?r2]? )
  FDUP F=  FDUP F= AND  0= ;
```

The logical negations N<, N=, N>, and N? can be expressed with the Forth phrases F< 0=, etc.

Forth words for the <= and >= predicates can be defined as

```
: F2DUP  ( f: r1 r2 -- r1 r2 r1 r2 )       FOVER FOVER ;
: F<=    ( f: r1 r2 -- ) ( -- [r1<=r2]? )  F2DUP F< F= OR ;
: F>=    ( f: r1 r2 -- ) ( -- [r1>=r2]? )  F2DUP F> F= OR ;
```

The <? and >? predicates can be expressed by the Forth phrases "F>= 0=" and "F<= 0=".[1]

Thus all twelve IEEE predicates can be expressed with the five required words, and as few as two.

---

1 It can be shown that the closure of the four fundamental IEEE comparison predicates under AND, OR, and negation consists of sixteen independent relations, including the twelve that IEEE requires. Two additional elements are the trivial, identically true and false relations, and the other two are "less than or greater than" and its negation, "unordered or equal". The five words of this word set, plus their five negations, implement the only nontrivial, transitive relations among the sixteen.

On the other hand, several current cpu's have efficient operations for all four of the fundamental IEEE comparisons, <, >, =, and ?. Low-level implementations of at least F<, F>, and F=, would be natural for such systems.

All of the words in the F0< family can be defined by analogy to

```
: F0<  ( f: r -- ) ( -- [r<0]? )  0E F< ;
```