Proposed Floating Point Environmental Inquiries in FORTRAN

W. Kahan, J. Demmel, J. T. Coonen
University of California at Berkeley

This is a proposal for floating point environmental inquiries in Fortran. The authors present it to the ANSI X3J3 Fortran Standards Committee on behalf of IEEE Working Groups 754 and 854, which are developing binary and decimal standards for floating point arithmetic. Although it is intended for inclusion in the next Fortran standard, so-called Fortran 8X, the scheme is designed to be compatible with Fortran 77 implementations.

1. Portability

Fortran is usually associated with high speed computation on mainframes and minicomputers. And numerical Fortran codes are considered portable when they behave reasonably across this class of machines. Portability has been achieved by defining parameters that demarcate the boundaries of the various machines' arithmetics. The Bell Labs PORT Library [4] is just one significant effort. More recently, W. S. Brown has devised a model of arithmetic [2] encompassing nearly all existing arithmetic engines. He captures their diversity in an abstract, parameterized machine which is in some sense the least common denominator of all existing machines. J. L. Blue's program [1] to compute the Euclidean norm of a vector examplifies the programming style that goes with Brown's model -- and the difficulty of writing such universally portable codes.

But the software situation is changing somewhat. Proposed IEEE standard P754 for binary floating point arithmetic [5] is gaining acceptance in the computing industry. For example, significant hardware support for the standard is already available from one microprocessor manufacturer (Intel) and is expected soon from several others. What is important is that these new processors will not be restricted to a few in-house systems. Rather, they will be embedded in computer systems marketed by diverse companies, and they will perform at the levels of today's minicomputers. The P754 proposal, and its decimal sequel P854, provide features lacking in most previous machines, features such as sticky exception flags for errors, a choice of responses to exceptions like over/underflow, and a choice of direction of rounding. To exploit these features programmers need access to them in high-level languages. And the means of access must be standardized for each language so that codes can, *with minimal extra effort*, be made portable across the entire family of "standard" systems.

2. Design Constraints

This proposal serves two rather different needs. Following the lead of others who have worked in this area, notably W. S. Brown, W. J. Cody, S. I. Feldman, B. Ford, and B. T. Smith, it provides access to machine parameters which permit programming in a style that defends against the peculiar ways machines handle roundoff and exceptions like over/underflow. This facilitates the first kind of portability above. On the other hand, the 754/854 proposals are recognized as important enough to warrant functions to access their features, even though those features are not universal.

The capabilities in this proposal are needed in Fortran 77 now. Therefore the proposal has been devised, particularly in its syntax, to be compatible with existing Fortran 77 systems. And, in order that the proposal be implementable at low cost on a broad range of Fortran engines, it has been designed to have negligible impact on compilers. For example, no new reserved words like .HUGE. are used. Instead, all inquiries are made through intrinsic functions in the same domain as mathematical functions like COS and TAN. This concentrates both the effort and the responsibility where they belong.

Ideally, an inquiry mechanism should be invisible to programmers not interested in it, and readily available to those who are. Since there is no simple "include" mechanism in Fortran 77, no convenient way exists to reserve a named COMMON area with numerous PARAMETERs and variables related to the environment. The prospect that programmers might enter the relevant

4. Huge and Tiny Numbers

Functions HUGE and TINY return floating point values near the limits of a machine's range, according to a string parameter FLAVOR.

```
FUNCTION HUGE( X, FLAVOR)
  real type X
  CHARACTER*6 FLAVOR
```

X is a dummy parameter whose value is ignored but whose format determines the format of the return value.

FLAVOR	return value
'MACH'	biggest ordered value, possibly +OV symbol or $+\infty$ (even though the machine may not permit the value to be used in subsequent comparisons)
'THRESH'	biggest finite value that can be used in or result from some arithmetic operations without triggering overflow, though it may behave anomalously in some other operations
'MODEL'	biggest number that can be used safely in Brown's model

Typically, the 'MACH' and 'THRESH' values would differ only on systems that support symbols for values outside the range of finite representable numbers. Some machines support signed ∞, or something very like it. Another possibility is an overflow symbol OV that stands for the interval strictly between ∞ and the largest finite representable number. 'THRESH' and 'MODEL' values would differ only when the Brown model penalizes the system some units of exponent range due to unseemly behavior. Three kinds of HUGE may seem extravagant at first sight, but the fact is that the corresponding return values from HUGE really do vary on some machines. The following table gives the parameter values for the double formats of three sample architectures.

	return value from HUGE(X, FLAVOR)		
FLAVOR	P754 double	VAX-11 D-format	Cray-1 double
'MACH'	$+\infty$	1.7×10^{38}	$+\infty$
'THRESH'	1.8×10^{308}	1.7×10^{38}	$2^{8191}\times(1-2^{-96}) \approx 5.4\times10^{2465}$
'MODEL'	1.8×10^{308}	1.7×10^{38}	$2^{8190}\times(1-2^{-94}) \approx 2.7\times10^{2465}$

```
FUNCTION TINY( X, FLAVOR)
  real type X
  CHARACTER*6 FLAVOR
```

As above, X is a dummy parameter whose value is ignored but whose format determines the format of the return value.

FLAVOR	return value
'MACH'	smallest positive value, possibly a +UN symbol or a denormalized number
'THRESH'	smallest positive value that can be used in or result from some arithmetic operations without triggering underflow, though it may behave anomalously in some other operations
'MODEL'	smallest positive number that can be used safely in Brown's model

This function is similar to HUGE. TINY(X, 'MACH') is the smallest representable positive value in the format of X. It could be a symbol, UN, that behaves arithmetically like the interval between 0 and the tiniest representable magnitude. On some systems, notably 754/854, TINY(X, 'MACH') is the smallest of a family of tiny numbers, beneath the stated underflow threshold, designed to make underflow gradual rather than abrupt. As above, the difference between the 'THRESH' and 'MODEL' values depends on the quality of arithmetic near the bottom of the exponent range. The following table gives the parameter values for the double formats of three

```
FUNCTION SCALB( X, FACTOR)
  real type X
  INTEGER FACTOR
```

It returns the value $X \times b^{FACTOR}$, where b is the machine radix. The parameter FACTOR is specified as an integer so that SCALB can be fast, since it is often used in inner loops. Even so, SCALB is expected to conform to system conventions for dealing with exponent over/underflow, which must not occur unless the final value lies out of range. Underflows are denormalized [5] on systems that underflow gradually; some systems underflow to zero or TINY(X, 'MACH'); some systems also set an error flag. Overflows may be set to HUGE(X, 'MACH') or, as is more usual, may stop computation altogether; there may also be an error flag. Note that because the INTEGER type may be much wider than the exponent field of X, severe over/underflow is possible.

8. Classification

The function CLASS returns an integer indicating the "character" of the floating point argument. This is helpful in filtering special operands.

```
INTEGER FUNCTION CLASS( X)
  real type X
```

The sign of the returned integer indicates the sign of X, even if the sign has no relevance (such as the sign of 0, usually taken to be +, on systems with no −0, or the sign of NaNs). The magnitude of the returned value is defined from the table:

magnitude	X
1	zero
2	finite, nonzero, normalized number
3	∞
4	denormalized number, a la 754/854 proposals
5	unnormalized number, possibly with zero significand
6	nontrapping NaN -- propagates without exceptions
7	trapping NaN -- triggers exception on attempted use
8	UN symbol, or numbers between TINY(X, 'MODEL') and TINY(X,'MACH')
9	OV symbol, or numbers between HUGE(X, 'MODEL') and HUGE(X,'MACH')
...	...

The arbitrary breakdown above is intended to facilitate branching with a case statement (computed GOTO in Fortran), or the IF-THEN alternative. The commonest cases appear at the top of the list. Although specified for a wide class of numeric entities, a particular implementation of CLASS will return only the values pertinent for the given machine.

9. Exception Flags

Some arithmetics, in particular 754/854, provide flags which are set when the corresponding floating point exception arises, and which are cleared only at the program's request. The function FLAG gives a programmer access to such flags. It returns the current setting of the flag, and allows the programmer the option of altering the flag. Thus, the exception flags appear to the programmer as implicitly defined global variables, although they can be accessed only through the function FLAG.

```
INTEGER FUNCTION FLAG( TYPE, VALUE)
  CHARACTER*6 TYPE
  INTEGER VALUE
```

where TYPE is one of

11. Relation to Brown's Model

This section relates the environmental inquiries presented here with those Brown and Feldman proposed [3] in connection with Brown's model. On a machine of radix b, Brown considers a system of *model numbers* consisting of zero and all numbers of the form

$$x = fb^e$$

where

$$f = \pm(f_1 b^{-1} + \cdots + f_p b^{-p}), \quad f_1 = 1, \cdots, b-1,$$

$$f_2, \cdots, f_p = 0, \cdots, b-1,$$

and

$$e_{\min} \le e \le e_{\max}.$$

The following table gives the model parameters and the computational procedures of Brown and Feldman in terms of the present proposal.

```
          RADIX = INT ( SCALB ( 1.0, 1 ) )  ... b
   MODELEPSILON = NEXTM ( 1.0, 2.0 ) - 1.0 ...maximum relative spacing in model
      PRECISION = 1 - INT ( LOGB ( NEXT ( 1.0, 2.0 ) - 1.0 ) )  ...minimum number
                  of radix-b digits
MODELPRECISION = 1 - INT ( LOGB ( MODELEPSILON ) )  ...minimum number of
                  radix-b digits, including a possible penalty if rounding is strange
     MODELHUGE = HUGE ( X, 'MODEL' )  ...biggest number in Brown's model, includ-
                  ing a possible penalty if overflow is strange
          EMAX = INT ( LOGB ( MODELHUGE ) ) + 1  ...biggest exponent
     MODELTINY = TINY ( X, 'MODEL' )  ...smallest number in Brown's model, includ-
                  ing a possible penalty if underflow is strange
          EMIN = INT ( LOGB ( MODELTINY ) ) + 1  ...smallest exponent
   exponent(X) = INT ( LOGB ( X ) ) + 1
    scale(X, E) = SCALB ( X, E )
   fraction(X) = SCALB ( X, -exponent( X ) )
synthesize(X, E) = SCALB ( fraction( X ), E )
        α(X) = synthesize( 1.0, max ( EMIN, 1 - MODELPRECISION + exponent(
                  X ) ) )  ...if X is nonzero
             = MODELTINY  ...if X is 0
        β(X) = synthesize( ABS ( X ), MODELPRECISION )
```

12. References

[1] Blue, J.L., "A Portable Fortran Program to Find the Euclidean Norm of a Vector," *ACM Trans. Math. Soft.*, 4, 1, March 1978, 15-23.

[2] Brown, W.S., "A Simple but Realistic Model of Floating-Point Computation," *ACM Trans. Math. Soft.*, 7, 4, December 1981, 445-480.

[3] Brown, W.S. and S.I. Feldman, "Environment Parameters and Basic Functions for Floating-Point Computation," *ACM Trans. Math. Soft.*, 6, 4, December 1980, 510-523.

[4] Fox, P.A., A.D. Hall, and N.L. Schryer, "The PORT Mathematical Subroutine Library," *ACM Trans. Math. Soft.*, 4, 2, June 1978, 104-126.

[5] "A Proposed Standard for Binary Floating Point Arithmetic," and supporting articles, *Computer*, 13, 3, March 1981, 51-87.