Accredited Standards Committee
X3 , Information Processing Systems
operating under the procedures of the
American National Standards Institute

doc. no .: X3T2/87-112
Date: 87-10-27
project:
reply to.: Ned Anderson
or Mary Payne
Digital, I-ILO2—3/M08
7? Reed Road Hudson, MA
01749 617-568-5823

Subject: Draft Proposal for a Language—Based Arithmetic Standard

Acknowledgements: Much of the material in the Foreword has been developed in a series of conversations with Brian Wichmann of the National Physical Laboratory in England.

The format and organization of the specifications have been adapted from that of the IEEE Standard 854 for floating point arithmetic.

****

Foreword

This foreword and the footnotes are not part of the draft standard.

The purpose of this standard is to help the scientific community, using many different languages and systems from many different vendors, to share their existing and future software. Some of the current difficulties in such exchanges involve

1. The absence of arithmetic specifications in many language standards*, e.g.
   (a) Accuracy of the basic arithmetic operations.
   (b) Handling of arithmetic exceptions.
2. The large number of de facto arithmetic standards implied by the arithmetic specifications of a large number of vendors. For the most part, the differences among vendors are relatively minor. Hence, this standard will build on the common elements of these de facto standards.
3. Problems in the interchange of binary data , which should be resolved by current efforts to standardize such data transfers.

* Thus, this standard will provide a common supplement to language standards, which could be incorporated by reference.

The following objectives, in order of priority, are designed to meet. the purpose of this standard.

1. The standard shall not conflict with the requirements of the standards for the main programming languages {e.g, Fortran, Ada, Pascal, PL/I, etc.).
2. The standard shall not include features which cannot be exploited within the main programming language standards.
3. The standard shall be formulated in such as way as to permit the majority of existing computer systems to claim conformance to the standard.
4. The standard shall be formulated in such a way as to allow efficient conformance by future high—performance systems using pipelining and concurrency.
5. To an extent consistent with the above constraints, the standard shall require the arithmetic properties needed by numerical analysts to deduce the correctness of conventional algorithms.
6. The standard shall consist of the minimum unambiguous specifications needed
   (a) By implementors to conform to the above requirements.
   (b) To allow rigorous conformance testing of processors.
7. The standard shall not conflict with data interchange standards.

There is no expectation that the standard will imply bit—for—bit compatibility among results obtained for the same program executed on systems from different vendors.

In addition to the de facto arithmetic standards, mentioned above, there are currently two IEEE Floating Point Standards, 754 and 854. There is also an international IEC standard, based on an early draft of IEEE 754. These standards deal primarily with specifications of arithmetic properties useful to numerical analysts [objective 5 above}. Little attention was given to the first four objectives. In particular, the rounding modes, the exception handling features and the concept of "unordered" are not supported by existing language standards, nor are they implemented by most current hardware.

Thus, the proposed standard can be regarded as a generalization of the IEEE standards to facilitate the interchange of scientific software. It is expected that the proposed standard will include both IEBE standards as conforming implementations, by virtue of Objective 3.

Contents

1. Definitions and Notation

1.1 Definitions.

Destination. The location for the result of a binary or unary operation. A destination may be either explicitly designated by the user or implicitly supplied by the system (e.g., intermediate results in subexpressions or arguments for procedures). Some languages place the results of intermediate calculations in destinations beyond the user's control. Nonetheless, this standard defines the result of the operation in terms of that destination's precision as well as the operands' values.

Exact. An operation is exact if its full true result is exactly representable in the range—precision of the destination.

Exponent. The component of a floating—point number that signifies the integer power to which the radix is raised in determining the value of the represented number.

Field. A digit string provided for a component of a floating point number.

Floating—point exception. This standard recognizes three types of floating-point exceptions which may occur as the result of an operation on floating—point operands: they are floating overflow, floating underflow, and divide by zero.

Fraction. The component of a floating point number that signifies the fractional factor in its mathematical representation.

Integer exception. The only integer exception relevant to this standard is integer overflow.

Non—Zero Floating-Point Number. A number which has a mathematical representation of the form

```
(S)(r**E)*F
```

where S, r, E and F are the sign, radix, exponent and significand (or fraction] components, respectively.

Normalized number. A non—zero floating point number with its fraction component in the half-open interval [1/r,1).

Operands. The source operands and destination of an arithmetic operation.

Radix. The base for the representation of floating—point numbers.

Shall. The use of the word "shall" signifies that which is obligatory in any conforming implementation.

Should. The use of the word "should" signifies that which is strongly recommended as being in keeping with the intent of the standard, although architectural or other constraints beyond the scope of this standard may on occasion render the recommendations impractical.

Significant. Synonym for the fractional component of a floating—point number.

Source Operand(s). The operand to which a unary operator is applied to produce a result, or the operands to which a binary operator is applied to produce a result.

Trap. A hardware transfer of control to system software.

ULP. Unit in the last place, that is in the least significant digit of the fraction or significand component.

User. Any person, hardware, or program not itself specified by this standard, having access to and controlling those operations of the programming environment specified in this standard.

Zero. An implementation specific digit string, used to identify zero as the value of a floating point number.

1.2 Notation

1. Arithmetic Operators. The arithmetic operator notation +, -, *, /, and ** is used to denote, respectively, addition;_subtraction, multiplication, division, and exponentiation.
2. Relational Operators. The relational operator notation <, <=, =, <>, >, and >= is used to indicate, respectively, less than, less than or equal, equal, not equal, greater than, and greater than or equal.

2. Scope

2.1 Implementation Objectives.

This standard is a functional specification. It does not specify implementation: In particular it does not specify whether the Standard is implemented in hardware, software, or a combination of the two. However, the intent of the Standard is that it be readily implementable in hardware. Also the intent is that the Standard be independent of any particular computer architecture, and that it apply to both scalar and vector processing.

2.2 Inclusions.

This standard specifies:

1. Constraints on the parameters defining the values of floating point numbers at two levels of re1nge—precision: "short"and "long".

2. The accuracy of the four basic floating point operations*, addition, subtraction, multiplication and division whose source operands and destination are all at the same level of range—precision.
3. Conversions between integer and floating point formats.
4. Conversions between short and long floating point range~precisions.
5. Floating point exceptions and integer overflow, and the actions accompanying their occurrence.
6. The provision of implementation—specific data to users.

2.3 Exclusions.

This Standard does not specify

1. Formats for internal storage of floating—point numbers.
2. The bias, if any, to be applied to the exponent field of a floating point number.
3. Representation and storage format of integers.
4. Results of all operations on integers, except conversions between floating point numbers and integers.
5. Operations whose operands are specified at different levels of range—precision or with different radices.

3. Precisions

This standard defines two floating—point ranqe~precisions: short and long.

The standard does not specify how to encode numbers for internal storage.

3.1 Floating Point Parameters.

3.1.1 An implementor shall provide a representation of floating point zero at each implemented level of range—precision. The representation(s) shall be implementation dependent.

****

* Other functions, such as SORT, and the REM and INT of the IEEE Standards are intrinsic functions for some, but not all, languages. All are readily implementable in software, but this standard does not include specifications for their accuracy.

3.1.2 For non—zero p—digit floating point numbers, four integer parameters specify each level of range-precision:

r —— the radix;

p —— the number of radix—r digits in the fraction;

EMAX —— the maximum exponent, and

EMIN —— the minimum exponent.

The parameters are subject to the following constraints:

1. Any integer between EMAX and EHIN, inclusive, is an in—range exponent for a floating point number.
2. r shall be the same for each level of rang-e—precision.
3. EMAX > r**(N — 1), where N is the number of radix—r digits accomodated in the longest supported integer format.
4. r >= 2 and p >= 2
5. EMIN <= 2 — 2*p and EMAX >= 2*p-1 to guarantee a minimal range for a given precision.
6. 2*EMIN + EMAX <= 3 - p
7. EMIN + 2*EMAX >= p + 1

The last four constraints follow the work of W. S. Brown*.

3.2 Short and Long Precisions

3.2.1 Short Precision: If an implementation provides only one level of range-precision, that level shall be identified as "short precision."

Otherwise, the lowest range—precision supported shall be called short precision. When necessary to distinguish from other parameters, those defining short precision are denoted by

EMAXS, EMINS, pS

3.2.2 Long Precision: When a second higher level of range—precision is supported, it shall be called "long precision." When necessary to distinguish from other parameters, those defining long precision are denoted by

****

* W. S. Brown, "A Simple but Realistic Model of Floating Point Computation," ACE Transactions on Mathematical Software Vol. 7, No. 4, December 1981. This paper contains a detailed justification of constraints 6 and 7. All computers in common use today satisfy these constraints by a comfortable margin.

EMRXL, EMINL, pL.

In addition to the requirements specified in Sec. 3.1, parameters for long precision shall be further constrained as follows:

pL >= 2*pS

EMAXL >= EMAXS

EIMINL <= EMINS

and EMAXL and EIMINL should satisfy the stronger inequalities:

EMAXL >= 2*EIMAXS

EMINL <= 2*EMINS

4. Operations

All conforming implemnentations of this standard shall provide a scalar processor for which the operations listed in Sections 4.1 through 4.4 shall be implemented with the source and destination operands all at the same level of precision*.

A system, which provides a vector processor shall provide vector operations satisfying the same specifications in such a way that its scalar and vector operations yield digit—for—digit compatible results.

4.1 Basic: Arithmetic Operations.

An implementation of this standard shall provide addition, subtraction, multiplication and division at each supported level of range-precision. All results shall be rounded as specified in Sec. 5.

These operations shall preserve the following arithmetic identities:

A*B = B*A

A + B = B + A

A - B + -(B - A)

4.2 F1oating—point precision conversions.

****

* Operations involving operands at more than one level of range—precision are readily implemented in software by making use of the conversion operations in Sec 4.2. Such operations are not specified in this standard, although they may be implicit in the rules for expression evaluation associated with a language standard.

It shall be possible to convert floating—point numbers between all supported levels of range—precision. If the conversion is to a lower range—precision the result shall be rounded as specified in Sec. 5. If the conversion is to a higher range—precision, the result shall be exact.

4.3 Floating Point, Integer Conversions.

It shall be possible to convert between all supported floating—point range-precisions and all supported integer precisions. The rounding algorithm for conversion to integer shall be language and implementation dependent.

4.4 Comparison.

It shall be possible to compare two floating—point numbers, without occurrence of floating overflow or underflow. Moreover, the result of a comparison shall be exact* ,

5. Rounding

Rounding takes a number regarded as infinitely precise** and, if necessary, approximates it to fit the destination's fraction field.

The following rounding requirements guarantee that non—zero results are strictly accurate to within one ULP. Zero results are either exact, or are a consequence of underflow [see Sec. 6.1).

Let R be the infinite range—precision result of an operation. If R = O, the destination shall be zero.

If R is not zero, then ignoring the constraints on the exponent field, let R1 and R2 be the two normalized p—digit numbers most closely bracketing R. R1 will be the same as R2, if the infinite precision fraction of R is exactly representable at the desired level of precision of the result. In any case, either R1 or R2 is now selected as a tentative result, the choice being implementation dependent.

Having tentatively selected R1 or R2, its exponent field is examined:

- If the exponent of the tentative result is in—range, the tentative result is stored in the destination.

****

* Thus, if a comparison is made via a subtraction, the constraints on the exponent field must be ignored, and the subtraction must be so implemented that an alignment shift does not discard the effects of any of the low order digits.

** It is never necessary to calculate an infinite range—precision result. However, a few guard digits are usually necessary to meet the requirements of this section.

- Underflow occurs if the exponent of the tentative result is less than EMIN. The default action to be taken for this case is to store-zero in the destination. See, however, Section 6.1.
- Overflow occurs if the exponent of the tentative result exceeds EMAX. See Section 6.1.

The algorithm for the choice between R1 and R2 shall be such that the same operation on the same source operands always yields the same result. Hence, if an implementation chooses to implement more than one rounding algorithm, a default must be identified as associated with this standard.

6. Standard Exceptions

6.1 Floating Point Exceptions

A implementation shall recognize three floating point exceptions: division by zero, floating overflow, and floating underflow. Of these:

1. Division by zero occurs only for the floating divide operation.
2. Floating overflow and floating underflow can occur on any of the four basic arithmetic operations, and also on conversion from long to short precision.
3. Conversion from short to long precision and comparisons shall never produce exceptions and shall always be exact.
4. Conversions from integer to a floating point format shall never produce an exception by virtue of constraint 2 in Sec. 3.1.2.

An enable/disable capability for underflow shall be provided. Disabled shall be the default, in which case zero shall be returned to the destination, and execution continued.

A trap shall be taken to system Software on division by zero, overflow and enabled underflow. The response of system software to these traps shall be language and implementation dependent. If the system provides access to user exception handlers, such access is governed by the specifications of Section 6.3.

6.2 Integer Exceptions

This standard is concerned only with integer overflow, which can occur on conversions from floating point to integer format.

An enable/disable capability for integer overflow shall be provided. Disabled shall be the default, in which case those low digits, which can be accomodated in the destination shall be returned, and execution continued.

A trap shall be taken for enabled integer overflow. The response of system software to this trap shall be language and implementation dependent. If the system provides access to user exception handlers, such access is governed by the specifications of Section 6.3.

6.3 Traps

All trapping shall be to system software, whose actions are language and implementation dependent.

A further transfer of control to user exception handlers shall be supported in accordance with the requirements of language standards, e.g. Ada and PL/I.

7. Implementation Specific Characteristics

An implementation shall provide means for a user to determine the following implementation specific information. Moreover, it shall make known to the user how this information may be obtained.

1. Whether the system implements vector operations.
2. The levels of range-precision supported.
3. Information on the internal representation of floating point numbers at each supported level of range-precision:
   (a) Internal representation of zero.
   (b) For non—zero floating point numbers, the values of the parameters of Sec. 3.1.2: r, p, EMAX, and EMIN.
   (c) The algorithm used for the selection of R1 and R2 (in Section 5) as the rounded result.
   (d) Lengths of the sign, exponent and fraction fields.
   (e) Storage length of the representation.
4. Details on the exception handling support provided by the system.
5. Information on other (implementation specific) exceptions, which could affect users' programs*.

****

* For example, the reserved operand exception on a VAX.