To: WG11 Participants
From: Editors of Language Independent Arithmetic, Part 1
Title: Response to IFIP WG2.5 Comments

Attached is a draft response to late public review comments on SO/IEC 10967:1991 Language Compatible Arithmetic received from IFIP Working Group 2.5 (Numerical Software). Please review the draft so that a final response can be approved at the upcoming meeting.

Response to Comments by Members
of
IFIP Working Group 2.5 (Numerical Software)
on
Version 3.1 of Draft ISO/IEC 10967:1991
Language Compatible Arithmetic

(DRAFT)

The comments contained in document SC22/WG11/N303 from IFIP Working Group 2.5 on Part 1 of Language Independent Arithmetic (formerly know as LCAS, now LIA-1) raise a number of issues. Many of these issues have been raised much earlier elsewhere and have resulted in a number of proposed changes to the LIA-1. In particular, the LIA-1 with proposed changes requires little more than is required by IEEE 754 and makes it very easy for a system using IEEE 754 arithmetic to conform to the LIA-1. The members of IFIP Working Group 2.5 are urged to review the proposed changes which are detailed in document WG11/N292 and a further revision WG11/N302. In what follows we have summarized and responded to the key points in the comments.

The Requirements of LIA-1 are Too Loose

The comments state that the proposed standard is too loose, but then complain that it is too tight to include Cray arithmetic and radix 10 systems. The proposed standard endeavors to be loose enough to cover the broadest possible range of existing and future computing systems, while remaining tight enough to insure predictable results across the range of systems covered.

It is a great disappointment that the arithmetic of systems as significant as those of CDC and Cray do not meet the LIA-1 requirements. The focus of the LIA-1 is on the users' needs rather than on the system implementations. Therefore, it is not the goal to coerce CDC or Cray into abandoning their arithmetic -- their users like it.

Systems using radix 10 arithmetic would indeed comply with the LIA-1. However, we are not aware of any such systems which would benefit from a standard aimed at portable software.

There is a suggestion that LIA-1 should be tightened to exclude VAX D-format and IBM double precision because their ranges are felt to be too restrictive. Whether or not the range of a particular floating point datatype is too restrictive depends very much upon the application in which it is used, and a large number of users have found that the ranges of the datatypes mentioned are adequate to meet the needs of their applications.

IEEE 754 Provides All the Portability That is Needed

The comments state that objectives of IEEE 754 are software that is robust and executes rapidly. There is no mention of portability. The implication is that portability will be achieved when manufacturers abandon their noncompliant architectures and users rewrite their applications. This may be viable as a long term goal. It would certainly make portability simpler, although it would not completely solve the problem. However, it is not very practical in the short term. Users and manufacturers have too large an economic investment in the systems and software that they would be asked to abandon.

There is a further implication that anything, such as the LIA-1, which would provide relief from the pressure to conform to IEEE 754 will detract from widespread acceptance of IEEE 754. That is a short-sighted view. Since the current proposals make it easy for a system using the IEEE 754 architecture to

meet the LIA-1 requirements, compliance with the LIA-1 can be seen as a safe intermediate step which might, in fact, help speed widespread acceptance of IEEE 754. Software which is currently LIA-1 compliant could be suitably used on a wide variety of current systems, and easily be adapted to take advantage of the IEEE 754 architecture if that does indeed predominate in the future.

LIA-1 Exception Handling Leads to Verbose Code

The code for exception handling is no more verbose than that which would be required by IEEE 754. Indeed, it will be less verbose in the cases where there is a difference.

If the programmer chooses not to handle exceptions, then there is no additional code required under either standard. However, if the programmer has unwisely chosen not to handle exceptions and one does occur, the LIA-1 notification requirements assure that the user of the program is not left with the impression that the program executed normally.

If the programmer does choose to handle exceptions and is programming in a standard language whose syntax supports the writing of user defined exception handlers, then the code needed should be identical on any implementation which conforms to the language standard. No more code would be required by LIA-1 than by IEEE 754.

If the programmer does choose to handle exceptions, but is programming in a language without exception handling syntax, then the programmer can use the status flags required by IEEE 754. However, program access to the flags is not specified by IEEE 754, and is likely to vary from one compiler to another. Thus a portable application will require code for all possible compilers, even among IEEE 754 machines. The datatypes and library routines for exception handling required by LIA-1 provide uniform program access not only to the status flags for IEEE 754, but to exception handling mechanisms available under other conforming architectures as well. In this case less code may be required by LIA-1, yet support portability across a wider range of implementations.

The example to which the comments allude displaying verbosity was provided, not by Kahan, but by the LIA-1 authors at a very early stage in the development. The comment would have been useful if it had been more timely, but with the current draft it is no longer relevant.

LIA-1 Notification Will Slow Performance

The LIA-1 section dealing with notification has been revised and in fact includes provisions similar to those suggested in the comments. The alteration of control flow need not be immediate. Since the alternative for the change of control flow for notification is only available through languages such as ADA, PL/1 or Lisp which have syntax permitting users to write their own exception handling code, the question of how long the alteration of control flow can be delayed without compromising the semantics of the language is left to the language standard.

The prospect of "millions of lines of output messages" resulting from arithmetic exceptions is no longer an option under the current proposal for notification. However, even that prospect is certainly more appealing than the alternative of having such a poorly written program complete without letting the user know that anything unusual happened.

LIA-1 Defeats Mathematical Intuition

The comments again point out that the computation

u := x / sqrt(sqr(x) + sqr(y)),

which can be "reasonably expected" to produce values of magnitude less than or equal to one, will produce a value greater than one under some conditions allowed by LIA-1. In fact the tightness of IEEE 754 is no help as shown in the following two paragraphs.

If all the computation is performed at working precision, then underflow can occur in either invocation of the function sqr. There are both single and double precision values such that sqr(y) underflows to zero, and sqr(x) is denormalized losing the low bits of the radicand, which produce u values greater than one on a variety of processors using IEEE 754 arithmetic. For example, in single precision

x = 5.62482753427186400000000000e-23
y = 1.32348898008484430000000000e-23
u = 1.06249988079071040000000000e+00 ,

or in double precision

x = 2.50060359317071170000000000e-162
y = 2.40991986510288410000000000e-181
u = 1.12499999999999980000000000e+00 .

If both sqr(x) and sqr(y) underflow to zero, u is assigned either plus or minus infinity.

Moreover, the underflow is difficult to anticipate since the intermediate calculations can give quite different results. The functions sqr and sqrt could be computed to extended or double precision, and all that one can say about the operator + is that it is likely to be computed with precision greater than single. These variations actually arise on a Sun/3 system with different compilers and compiling options.

This example brings up another very important issue which is the distinction between theoretical mathematics and computer mathematics. Since computer mathematics is only an approximation to theoretical mathematics, there will always be some theoretically based expectations that are not met by the computer approximation. The only properties which one can "reasonably expect" from a computer approximation are those which are stated explicitly in the specifications of a standard, or which can be derived from those specifications.