TABLE OF REPLIES /1994-04-13/ TABLEAU DES REPONSES

ISO/IEC DIS 10967-1
JTC 1/SC 22/WG 11
VOTING BEGINS ON/DEBUT DU VOTE: 1993-09-30
TIME LIMIT FOR REPLY/DELAII: 1994-03-30

TITLE: Information technology -- Language independent arithmetic
Part 1: Integer and floating point arithmetic

TITRE: Technologies de l'information -- Arithmétique de langage indépendant
Partie 1: Arithmétique de point en nombre entier et flottant (DIS
distribuée en version anglaise seulement)

<table>
<thead>
<tr>
<th>MEMBER BODY/COMITE MEMBRE</th>
<th>DISAPPROVAL/DESAPPROBATION APPROVAL/APPROBATION</th>
<th>ABSTENTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Australia (SAA)</td>
<td>P</td>
<td>I</td>
</tr>
<tr>
<td>Austria (ON)</td>
<td>P</td>
<td>I</td>
</tr>
<tr>
<td>Belgium (IBN)</td>
<td>P</td>
<td>I</td>
</tr>
<tr>
<td>Brazil (ABNT)</td>
<td>P</td>
<td>I</td>
</tr>
<tr>
<td>Canada (SCC)</td>
<td>P</td>
<td>I</td>
</tr>
<tr>
<td>China (CSBTS)</td>
<td>P</td>
<td>I</td>
</tr>
<tr>
<td>Czech Republic (COSMT)</td>
<td>O</td>
<td>I</td>
</tr>
<tr>
<td>Denmark (DS)</td>
<td>P</td>
<td>I</td>
</tr>
<tr>
<td>Egypt (EOS)</td>
<td>P</td>
<td>I</td>
</tr>
<tr>
<td>Finland (SFS)</td>
<td>P</td>
<td>I</td>
</tr>
<tr>
<td>France (AFNOR)</td>
<td>P</td>
<td>I</td>
</tr>
<tr>
<td>Germany (DIN)</td>
<td>P</td>
<td>I</td>
</tr>
<tr>
<td>Ireland (NSAI)</td>
<td>P</td>
<td>I</td>
</tr>
<tr>
<td>Italy (UNI)</td>
<td>P</td>
<td>I</td>
</tr>
<tr>
<td>Japan (JISC)</td>
<td>P</td>
<td>I</td>
</tr>
</tbody>
</table>

** = Comments/Commentaires

P = P-member having abstained and therefore not counted in the vote/
Membre (P) s'abstenant de voter; n'est donc pas compté dans le vote

TOTAL 21 2

P-MEMBERS VOTING:
MEMBERS (P) VOTANT: IN FAVOUR OUT OF REQUIREMENT
18 20 = 90% >= 66.7% CRITERE

MEMBER BODIES VOTING:
COMITES MEMBRES VOTANT: NEGATIVE VOTES OUT OF REQUIREMENT
2 23 = 9% <= 25% CRITERE

THIS DRAFT HAS THEREFORE BEEN APPROVED in accordance with the
ISO/IEC Directives, Part 1, subclause 2.6.3.

CE PROJET EST DONC APPROVE selon les Directives ISO/CEI,
Partie 1, paragraphe 2.6.3.
AFNOR's comments relating to the disapproval vote on DIS 10967-1 "Information Technology - Language independant arithmetic - Integer and floating point arithmetic" (ISO/IEC JTC 1/SC 22)

AFNOR thinks that this DIS has a very good technical content and shall progress to IS stage. Nevertheless AFNOR disapproves DIS 10967 mainly because the French Title is fully incorrect.

1. French Title (Major comment) : it shall be: "Technologies de l'information - Arithmétique indépendante des languages - Arithmétique des nombres entiers et en virgule flottante".

2. Other comments
   - P. 83 : the NOTE located just before E7 is obsolete since ISO/IEC 1539 : 91 (FORTRAN 90) has been published.
   - Annex G example programs
     - Some mistakes have been found and shall be corrected :
       - p.91 G1 : the second line of the first formula should be read : print 3, 'this platform has insufficient precision.'
       - The second formula : if [(HUGE(...) ... is not portable.
       - p.92 G3 : Ada example of program
         Assignment statements in Ada are denoted by the symbol := (example, second line :
         prev.approx := first.guess (input)
       - p.92 G4 : the example of FORTRAN code shall begin by call (ctrl indicators (....))
   - Annex H Bibliography

Once again, AFNOR would like to stress that the acceptance of the proposed modifications, in particular comment 1. will change its disapproval vote to approval.
To cast a vote on a draft International Standard, national bodies shall complete and sign this ballot paper, and return same with any comments to the ISO Central Secretariat.

All national bodies are invited to vote. P-members of the joint technical committee concerned have an obligation to vote.

☐ We approve the technical content of the draft as presented (editorial or other comments may be appended)

☐ We disapprove for the technical reasons stated at annex
  ☐ Acceptance of specified technical modifications will change our vote to approval

☒ We abstain (for reasons below)

Remarks:

For the time being, our national technical committee USM/TC INF, doesn't cover all the subcommittees of ISO/IEC JTC 1, yet.

signature  Technical secretary USM/TC INF

Janez Hočevar, dipl.ing.
Swedish comments on DIS 10967-1:1993
Comments on LIA-1 (v. 4.1)
[ISO/IEC DIS 10967-1:1993]
1994-03-14

Kent Karlsson
Department of Computing Science
Chalmers University of Technology and The University of Göteborg
S-412 96 Göteborg
Sweden
<kent@cs.chalmers.SE>

1 pages

1. Natural numbers (and monus)

1.1. Current situation in LIA-1 (v. 4.1)
LIA-1 currently allows both unbounded and unsigned integer types. Strangely enough it currently cannot consider integer types that are both unbounded and unsigned as conforming! That is, natural numbers are not covered. Natural numbers are not very commonly implemented, but it is still very strange of LIA-1 not to cover them, since it is the most basic number type.

1.2. Proposed changes
1. Add a boolean parameter for the integer types, signed. For natural numbers signed shall be false, and bounded shall also be false.
2. Add the monus operation for the integer types, defined as
   \[
   \text{monus}_I(x, y) = \max\{0, x - y\} \quad \text{if} \quad 0 \leq x \text{ and } 0 \leq y
   \]
   \[
   = \text{undefined} \quad \text{if} \quad x < 0 \text{ or } y < 0
   \]
3. Add the monus operation for the floating point types, with, in principle, the same definition.
   Monus is a useful operation on strictly non-negative quantities. The monus operation could be optional.

2. "Modulo" integers

2.1. Current situation in LIA-1 (v. 4.1)
LIA-1 now allows so called "modulo" integers. They are ill-conceived and devoid of applications. They hide integer overflow exceptions in the most treacherous way (they are not signalled at all), and this goes against one of the goals of LIA: that exceptions should be "hard-to-ignore"! This means that the "modulo" integers do not support reliable computing, a major LIA goal.

2.2. Proposed changes
1. Remove the "modulo" integers, they are against the very spirit of LIA
3. $add_{F}$

3.1. Current situation in LIA-1 (v. 4.1)

As the LIA-1 v. 4.1 report says, ideally $add_{F}$ should be $+$. I see no real reason for allowing anything else. Removing $add_{F}$ would simplify LIA and make LIA more strict.

3.2. Proposed changes

1. Replace $add_{F}$ with $+$.

4. $div_{F}^{1}/rem_{F}^{1}/div_{F}^{2}/rem_{F}^{2}/mod_{F}^{1}/mod_{F}^{2}$

4.1. Current situation in LIA-1 (v. 4.1)

Two different versions of the pair of operations $div_{F}/rem_{F}$ are conforming. LIA-1 distinguishes between them, but far from sufficiently clearly. The distinction is made by using the superscripts "1" and "2". Firstly, this is not sufficiently mnemonic and even the authors get confused (in annex E). Secondly, these superscripts are sometimes omitted, adding to the confusion. The name $div_{F}$ is still used despite the future name problems with floating point flooring division.

The most important property that useful $div_{F}/rem_{F}$ functions must fulfill, translational invariance, is broken by the truncation towards zero variety (those with superscript "2"). There is no purpose in breaking the translational invariance property.

There are still two varieties of $mod_{F}$, distinguished by LIA-1. The same comments about the superscripts as mentioned above hold. The first variety of $mod_{F}$ is identical to the first variety of $rem_{F}$. There is no purpose in giving the same operation two names. The second variety of $mod_{F}$ is an unduly restricted version of the first variety. There is no purpose in having this second variety.

4.2. Proposed changes

1. Remove $div_{F}^{2}$, $rem_{F}^{2}$, $mod_{F}^{1}$, and $mod_{F}^{2}$. These varieties are pointless (or already present with another name), and should not be promoted by LIA-1.

2. Rename $div_{F}^{1}$ to $quot_{F}^{1}$, and rename $rem_{F}^{1}$ to $rem_{F}^{1}$ (or to $mod_{F}^{1}$). This way there is "name harmony" even when adding $quot_{F}^{2}$ (flooring floating point division), and the superscript 1 is mnemonic enough (even if just one letter) not to cause any confusion.

3. Add $quot_{F}^{2}$ and $rem_{F}^{2}$ (or use the name $mod_{F}^{2}$) as optional operations. These should be defined to be the "ceilinging" integer quotienting and remainder.

E.g. $century = quot_{F}^{1}(year, 100)$.

4. When referring to the quotient and remainder operations by name, in the LIA documents, the superscript should always be included.

5. Correct and change the bindings in annex E accordingly.

6. Introduce operations $quot_{F}^{1}/rem_{F}^{1}$ and, as optional operations, $quot_{F}^{2}$ and $rem_{F}^{2}$ with, in principle, the same definitions as in the integer case, but taking into consideration that they are floating point operations. Add bindings in annex E accordingly.
4.3. Replies to the official reply to my comments on version 4.0

"We have tried to clarify the two versions of div/rem allowed."

Yes, but not at all sufficiently. See above.

"One provides translational invariance (as you requested), and the other recognizes the Fortran behaviour that is wired into many existing machines. We agree that the former version is more desirable, but we cannot ignore existing practice."

What is existing practice? Existing practice among programmers is to regard quotient/remainder operations as being useful only when the arguments are positive. Why is that? There are two reasons:

1. In most (but not all) applications of quotient/remainder operations the arguments actually are positive. And for positive arguments flooring and truncating quotient/remainder coincides.

2. This has lead to the mistaken choice of truncating quotient/remainder operations in Fortran and later (original) Pascal and Modula. And in later Pascal and Modula to the likewise mistaken restriction to only positive second arguments to their remainder functions (by the name of mod). In C one has done an even greater mistake in leaving it unspecified which remainder operation is denoted by its remainder operator (%); it's up to the implementation. Dismally, LIA-1 has followed C's approach, which, quite unnecessarily, makes the quotient/remainder operations useless on negative arguments.

This, however, is no reason to continue promoting quotient/remainder operations that are ill-defined or useless. On the contrary, LIA-1 is an excellent opportunity to clean up the mess! Many (all) applications of quotient/remainder operations naturally extend to negative first arguments, provided that it is a translationally invariant quotient/remainder operation that is used. The truncating variety is no good for any application. As for negative second arguments: direct applications seem few, but the sign laws give no reason to forbid negative second arguments.

If LIA-1 is to allow just any strange and ill-defined operations that happens to exist in Fortran, Modula, or C, then what is the purpose of LIA-1? Omitting ill-defined and ill-conceived operations from LIA-1 obviously does not forbid a language to have them, only that, if conformance with LIA-1 is sought, that the proper varieties must be provided. Hence there is no conflict with any existing programming language to only promote the properly defined and well-conceived varieties.

I'm not asking you to ignore existing practice. I'm asking you to recognize it! The truncating varieties are not supportive to reliable computing, so they go against one of the main goals of LIA, and should therefore not be in LIA! The problem is compounded by the sloppiness of distinction still present at this point in LIA-1.

5. \( CVT_{\mathbb{F}} \rightarrow \mathbb{I} \)

5.1. Current situation in LIA-1 (v. 4.1)

The \( CVT_{\mathbb{F}} \rightarrow \mathbb{I} \) operation is so loosely defined that it is useless.

5.2. Proposed changes

1. Remove the \( CVT_{\mathbb{F}} \rightarrow \mathbb{I} \) operation.

2. Add operations \( \text{ceiling}_{\mathbb{F}} \rightarrow \mathbb{I} \), \( \text{floor}_{\mathbb{F}} \rightarrow \mathbb{I} \), and \( \text{round}_{\mathbb{F}} \rightarrow \mathbb{I} \) with appropriate definitions, and require that the rounding style for \( \text{round}_{\mathbb{F}} \rightarrow \mathbb{I} \) be specified (halves-to-even or halves-to-odd).
UK COMMENT ON ISO/IEC DIS 10967-1

Editorial comment

4.1, line 10: for 'on sets' read 'on non-empty sets'.

CMJ
15 March 1994
UNITED STATES COMMENTS
ACCOMPANYING
RECOMMENDATION TO APPROVE

DIS 10967-1
"LANGUAGE INDEPENDENT ARITHMETIC - PART 1"
A.1 page vi, paragraph 6
Delete it. In fact it does not comply with US comment 1.1 to 2nd CD LIA-1.

A.2 page vi, paragraph 7
Delete the last sentence. It does not comply with US comment 1.1 to 2nd CD LIA-1.

A.3 page vi, last paragraph
Delete it.

A.4 page viii, second line
Replace "characterise" by "describe some of".

A.5 page viii, first paragraph
Second and third aim need to be better formulated.

A.6 page viii, third, fourth, fifth, sixth paragraphs of the benefits
The Benefits section should be reexamined and modified to ensure that it is accurate and does not create false impressions or contain false promises.

A.7 page 1, second paragraph
The last sentence should read: "Rather, this International Standard ensures that the properties of the arithmetic of conforming arithmetic types are made available to the user."

A.8 page 2, 9
Delete it. It contradicts with requirements when isc_559 is true.

A.9 page 2, last sentence
Delete it.

A.10 pages 4 and 5, "continuation values" and "exceptional values"
They need to be redefined properly, need clarifications, bringing in tune with the common usage and with the usage of "exceptional operands" and "exceptional results" in IEC 559.

A.11 page 5, "error", (3)
Should make it clear that except for those phrases, error and exception are not synonyms.

A.12 page 5, "exception"
Should get a better definition or be deleted.

A.13 page 5, "exceptional value"
See comments A.10 and A.11 about continuation value and exception.

A.14 page 5, "notification", second line
Replace "result in a notification" by "causes a notification".
A.18
page 5, "rounding"
Needs a meaningful definition.

A.18
page 6, "round to nearest"
Add text along these lines: "If the adjacent values are equidistant from v, either may be chosen according to precise rules which will be properly documented, and available at run time".

A.17
page 6, paragraphs 2, 3, 4 in section 5
They need to be changed in accordance with comments made above. This is a non-trivial editing task, beyond the scope of a comment or simple request for change.

A.18
page 7, first paragraph
Change it to "Whenever an arithmetic operation causes an exception, ...").

A.18
page 7, fourth paragraph (Note)
This should be elaborated on and placed in a more prominent place, since it is basically defining what implementation and conformance mean.

A.20
page 11, line after second Note
Delete "that spans all of R". The word span has a universally accepted mathematical meaning which is different than the one implied here.

A.31
page 12, section 5.2.2
This is at best misleading. Section 5.2.9 will define iec_559. In c-93 iec_559 is not set, these operations are required to have "larger" nature, and further differentiation between the exceptional values is required. Specifically, the section is OK as it stands when iec_559 is not set, but it needs to be much more specific when iec_559 is set.

A.22
sections 5.2.4, 5.2.5, 5.2.6, 5.2.7
US comment 5.4 to 2nd CD 1.1.1 referred to these and it was not implemented. The US acknowledges the difficulty the editor(s) is(are) facing in eliminating the helper functions, but they should be eliminated if the standard is to be useful.

Take the point of view of the user and of the compiler writer. From the point of view of the user, what he sees is an operation, e.g. an addition. This addition is close, but differing from the real addition. This user needs to know in what ways and how much it is differing, and what properties it has (e.g. what axioms it satisfies). From the point of view of the hardware manufacturer and compiler writer it is also impractical to have to satisfy axioms involving these helper functions. If the adders are appearing naturally through the algorithms involved in the design of the hardware/software system, everything is simple; otherwise there is no reasonable way they can make sure they are providing a conforming implementation. The standard should provide either precise algorithms or precise descriptions of the result. What matters for everybody is the properties of the operations visible to the user.
sections 3.2.4, 5.2.5, 5.2.6, 5.2.7

There is no point in going through detailed editing comments since comment A.22 just suggested that this part has to be completely rewritten. The purpose of this comment is to point out that one needs to be careful with the usage of words. For example "shall" and "should" is are reserved words in the language of the standard. Therefore they should not be used in the context of add and rmd and, which are not required functions.

Another example is rounding. It is "defined" for the second time at the beginning of 5.2.5, but this definition is not better than the first one. The standard should either assume that the readers of this standard know what rounding is, or give a serious definition.

The list of editing problems goes on: the note in 3.2.4 talks of requirements in connection with add which is not required. The last sentence of 5.2.3 is grammatically bad.

A.24

pe 17, section 5.2.8, Note
The reference is probably to A.5.2.8, not A.5.2.3.

A.25

page 17, fifth line from below
This is factually incorrect. The behavior is specified when iec_559 is set.

A.26

page 17, last three lines
The definition of rmd_style needs to also accommodate the case when iec_559 is set.

A.27

section 5.2.9
Section 5.2.9 is incomplete with respect to the relationship between LIA-1 and IEC 559. Both the overall relationship and the details need to be defined. Having IEC 559 normatively included by reference in LIA when the flag IEC 559 is true implies changing not only this section, but other sections as well.

A.28

page 19, last line
The meanings of "shall" and "distinct" are probably not the usual ones. In fact it is not clear what this sentence is meant to say.

A.29

section 6
The US has already called attention before to the need of rewriting this section, and this comment is made to call attention to it again. Also note that US comment 5.6 to 2nd CD LIA-1 was calling for substantial clarifications. While WG11 rejected US comment 5.6 to 2nd CD LIA-1, it was agreed that certain clarifications will be made. They were not made at all. It is not worth it going through the text of this section trying to improve the words. The whole section needs to be reworked. Without a clear understanding of what notification means and how it is to be done a major part of this standard becomes useless.

A.30

section 7
Section 7 is still too vague. It needs to be improved.
The current definition of integer negate is:

\[ f \_ I(x) = \begin{cases} \neg x & \text{if } -x \text{ in } I \\ \text{integer\_overflow} & \text{if } -x \text{ not in } I \end{cases} \]

The last line (integer\_overflow) should be changed to:

\[ = \text{wrap\_I}(-x) \quad \text{if } -x \text{ not in } I \text{ and modulo = true} \]
\[ = \text{integer\_overflow} \quad \text{if } -x \text{ not in } I \text{ and modulo = false} \]

The current definition of integer absolute value is:

\[ s \_ I(x) = \begin{cases} |x| & \text{if } |x| \text{ in } I \\ \text{integer\_overflow} & \text{if } |x| \text{ not in } I \end{cases} \]

The last line (integer\_overflow) should be changed to:

\[ = \text{wrap\_I}(|x|) \quad \text{if } |x| \text{ not in } I \text{ and modulo = true} \]
\[ = \text{integer\_overflow} \quad \text{if } |x| \text{ not in } I \text{ and modulo = false} \]

The current definition of integer to integer conversion is:

\[ t \_ (Ia \rightarrow Ib)(x) = x \quad \text{if } x \text{ in } Ib \]
\[ = \text{integer\_overflow} \quad \text{if } x \text{ not in } Ib \]

The last line (integer\_overflow) should be changed to:

\[ = \text{wrap\_Ib}(x) \quad \text{if } x \text{ not in } Ib \text{ and modulo\_Ib = true} \]
\[ = \text{integer\_overflow} \quad \text{if } x \text{ not in } Ib \text{ and modulo\_Ib = false} \]

where modulo\_Ib is the modulo parameter for type Ib.

The current definition of floating point to integer conversion is:

\[ r \_ (F \rightarrow I)(x) = \text{rnd\_}(F \rightarrow I)(x) \quad \text{if } \text{rnd\_}(F \rightarrow I)(x) \text{ in } I \]
\[ = \text{integer\_overflow} \quad \text{if } \text{rnd\_}(F \rightarrow I)(x) \text{ not in } I \]

The last line (integer\_overflow) should be changed to:

\[ = \text{wrap\_I}(\text{rnd\_}(F \rightarrow I)(x)) \quad \text{if } \text{rnd\_}(F \rightarrow I)(x) \text{ not in } I \text{ and modulo\_I = true} \]
\[ = \text{integer\_overflow} \quad \text{if } \text{rnd\_}(F \rightarrow I)(x) \text{ not in } I \text{ and modulo\_I = false} \]

where modulo\_I is the modulo parameter for type I.

Since unsigned int and unsigned long in C are now recognised as conforming types, the following C "cast" operations should be listed as possible convert to integer operations on page 73.

(\text{unsigned int}) x, \quad (\text{unsigned long}) x

In addition, the next paragraph should be reworded as follows:

The C standard requires that float to integer conversions round toward zero. A proper C binding for LIA-1 should either accept C's rounding requirements for these conversions (and use the cast notation) or provide separate LIA-1 conversion functions that round to nearest.
The code example in annex G.3 is not legal ADA (it uses = rather than =). Replace it by

```ada
Approx, Previous_Approx : Float;
N : constant Float := 5.0;

 Previous_Approx := First_Guess (input);
 Approx := Next_Guess (input, Previous_Approx);
 while abs(Approx - Previous_Approx) > N*LIA1.Unit_Last_Place(Approx) loop
   Previous_Approx := Approx;
   Approx := Next_Guess (input, Previous_Approx);
 end loop;
```

WG11 should consider adding the following clause to the Rationale:

4.5.1.4 Relations among integer types

An implementation may provide more than one integer type, and many
current systems do. When one modular integer type I is a subset of
another modular integer type J, it is desirable that the cardinality
(maxint-minint+1) of I be an even multiple of the cardinality of I.
This property insures that conversions between I and J preserve the
modular structure of these types. For example, if x + y = z in J,
then

\[ cvt(J\to I)(x) + cvt(J\to I)(y) = cvt(J\to I)(z) \]

in I.

In annex C, clarify that a binding standard for LIA-I *should* include
bindings for all optional IEC 559 features, even though not all
implementations of IEC 559 will provide those features.

John Reid of Rutherford Labs has suggested various improvements to the
La
t Fortran binding. John Klein in MIT has done the same for PL/I.
GI, should consider consulting with these experts to produce improved
versions of the example Fortran and PL/I binding.

The US feels that binding standards for LIA-I and IEC 559 are essential.
The US urges SC22/WG11 and other standards bodies to take whatever steps are
needed to ensure that such binding standards are developed and adopted as
soon as possible.