WG14/N389
X3T11/94-074

Accredited Standards Committee
X3, Information Processing Systems*

Date: 28 November 1994

Reply to: Craig Schaffert
Digital Equipment Corp
Cambridge Research Lab
One Kendall Square, Bldg 700
Cambridge, MA 02139

To: X3J Committee Chairs

Re: ISO/IEC DIS 10967-1
Language independent arithmetic –
Part 1: Integer and floating point arithmetic

Dear Committee Chair,

Your committee is listed as coordinating liaison for ANSI project 686-I (Language independent arithmetic, part 1) currently under development by X3T2 and ISO/IEC JTC1/SC22/WG11.

I am enclosing copies of three documents that you might find of interest:

1. Responses to U.S. Comments on ISO/IEC DIS 10967-1 (X3T2/93-080)
2. Proposed U.S. Position on ISO/IEC DIS 10967-1 (X3T2/93-008)
3. Resolution of International Comments on LIA-1 (SC22/WG11 N401)

The public review comments are collected in document X3/94-0048.

If you have any comments or questions on this document, please forward them to the X3T2 chair

John Sharp
Sandia National Laboratories
Org. 4400 – P.O.Box 5800
Albuquerque, NM 87185

Feel free to contact me as well.

Thank you for your help in this matter.

Sincerely,

Craig Schaffert

Craig Schaffert, for X3T2

019

RESPONSES TO U.S. COMMENTS ON ISO/IEC DIS 10967-1
Language independent arithmetic --
Part 1: Integer and floating point arithmetic

April 1994

The draft international standard version of ISO/IEC 10967-1 [1] (LIA-1) was reviewed in the U.S. from 12 November 1993 through 11 January 1994.

Comments from 9 individual reviewers were received, as well as one set of comments from an X3 liaison committee. All of these comments were considered in formulating the U.S. position on LIA-1 [3], and all are addressed here.

COMMENT #1: Peter Farkas (Sun Pro)

Comments on the standards process: Comments on the general standards development process should be directed to ANSI.

Comments on the goals of LIA-1: It is clear that not everyone agrees with the goals of LIA-1. However, the majority of X3T2 feels that the current goals are valid, and that the standard is valuable.

Specific proposals for changes: Virtually all of these proposals were adopted as part of the official U.S. comments [3]. Some comments were omitted because they contained no concrete suggestions.

COMMENT #2: Keith Bierman (Sun Pro)

Commentor's pages 1 thru 4, comments on the standards process: Comments on the general standards development process should be directed to ANSI.

Commentor's pages 5 and 6: It is clear that not everyone agrees with the goals of LIA-1. However, the majority of X3T2 feels that the current goals are valid, and that the standard is valuable.

Pages 7 thru the end: Since this is a verbatim repetition of your previous comments, please see our previous response to these comments.

COMMENT #3: Kenneth W. Dritz (Argonne National Lab)

Thank you for your kind words about the mathematical style chosen for LIA-1.

COMMENT #4: Robert Jervis (Sun Microsystems)

Comments on the goals and utility of LIA-1: It is clear that not everyone agrees with the goals of LIA-1. However, the majority of X3T2 feels that the current goals are valid, and that the standard is valuable.

IEC 559 is more than adequate: We disagree. See our previous comments on this issue, particularly <G8> and <G12> in [2].

Page 7, the arithmetic types: We agree that standard bindings are essential. See comment C1 in [3].

Page 8, modulo integers versus overflow: You express concern about the efficiency of multiply, divide, and remainder in Sparcs, but the LIA-1

definition is exactly equivalent to the definition in IEEE P1754 (which describes the Sparc architecture). You worry about other architectures, but haven't presented any machine in common use that cannot implement these operations with acceptable efficiency.

Page 23, relationship with language standards: Since both topics are properly addressed in language standards, we will retain the existing section.

COMMENT #5: Paul Eggert (Twin Sun Inc)

Introduction: The introduction to an ISO standard is an informal description of the standard's content and purpose. It is not supposed to be a proof of anything. The U.S. comments [3] do recommend eliminating claims that have been misunderstood.

Scope: Holm's approach was considered and excluded early on. Every approach has its problems, but the majority felt that a more precise definition of the basic floating point operations (one that allowed analysis of Dekker's algorithm, for example) was necessary.

As for Cray, the majority of participating countries think that excluding Cray is a good thing.

Clause 5.1.2: What existing machines in common use are excluded by the LIA-1 definitions?

Clause 5.1.3: We agree, see comments B1 and B2 in [3].

Clause 5.3: We agree, see comments B3 and B4 in [3].

Clause 6.1: Notification by exceptional values is allowed by the DIS, but the current text is clearly open to misunderstanding. The US comments [3] call for clarification in this area.

The programming language (or binding) standard can devise any method for notification whatsoever. If these standards don't specify a method, recording of indicators must be provided. Note that this does not preclude the simultaneous use of other methods such as exceptional values.

The reference to functional languages is part of an example, not a restriction.

Annex A.5.2.0.4: We can add the cross reference, but see no reason to soften the wording on NaNs and infinities.

Annex A.6.1.1: Notification by exceptional values is allowed under clause 2, not some convoluted reading of clause 6.

LIA-1 helps programmers know about rounding error (via parameters) and about exceptions (via notification). LIA-1 constrains systems to provide this information, but cannot force programmers to make use of it rationally. Note that programmers must take some positive action to suppress a notification completely.

Floating point arithmetic is intrinsically approximate (programmers who expect exact arithmetic should use integers or fixed point). There is a qualitative difference between rounding error (which programmers always have to account for) and the events that require notification such as dividing by zero or overflow.

Annex A.6.1.2: LIA-1 is compatible with solutions to this problem. Example solutions include Ada style exceptions, per-thread indicators, and returning exceptional values. It is better to point out the problem to language (and library) designers than to remain silent. Error handling is a general

problem applying to all programming, it is not confined to arithmetic processing.

The comment that "notifications do not get lost" is advice, not a requirement. Requirements never occur in informative annexes.

Annex C: We agree that binding standards for IEC 559 are important. See comment C1 in [3]. But we feel that effort should be put into official standards, not into examples. Procedurally, this can be done faster by working on each language separately, not by tying them all together.

Annex C.2: We agree.

Annex E: We disagree, reviewing such bindings for LIA-1 and then again for an official binding standard would only lead to delay. Other groups are already working on bindings to IEC 559, and it's best not to muddy the waters.

Annex E.1: Good point, but this is up to the official C binding.

Annex E.4, page 73: We agree.

Annex E.4, page 75: We agree, see [3].

Annex F.4: We agree.

Annex F.5: OK, but other reviewers have suggested removing the clause entirely.

Annex G: Examples are not proofs. They are intended to show uses of particular features, and must be short in order to be clear.

Annex G.1: We agree.

Annex G.3: These are terse examples, not intended to be complete in all areas.

Annex G.5: True, but the code is correctly quoted from the original authors. We will suggest noting that the original authors assume that exceptions will not arise.

Annex G.6: Again, we will suggest noting that exceptions are being ignored for clarity.

COMMENT #6: Russ Tuck (MasPar Computer Corp)

Russ Tuck's comments are in support of Keith Bierman's. See our response above.

COMMENT #7: James Gosling (First Person Inc)

James Gosling's comments are in support of Keith Bierman's. See our response above.

IEC 559 is more than adequate: We disagree. See our previous comments on this issue, particularly <G8> and <G12> in [2]. Note that LIA-1 is not a replacement for, or alternative to, IEC 559. LIA-1 addresses different issues.

COMMENT #8: Fred Tydeman (IBM)

Page vi: The modulo parameter seems to meet your needs. This suppresses overflow checking and gives a well-defined result. Suppressing overflow checking while leaving the result undefined was explicitly rejected by

several previous reviewers.

Page 2: We don't object, but enumerating all out-of-scope arithmetics could go on forever.

Page 3: We agree.

Page 10: OK.

Page 11: No, which types are "useful" is application specific.

Page 18: We disagree that readers will be confused.

Page 19, integer to integer conversions: We agree, see [3].

Page 19, conversions to floating point: No, the majority of reviewers so far have advocated some form of round to nearest for conversions to floating point, and binding standards are free to alter this.

Page 20: No, LIA-1's requirements apply to such programs. Do you mean that LIA-1 does not say anything specific about such programs or treat them differently? That's true.

Page 30: No, language standards seldom set up a validation process.

Page 58: Sine, and other mathematical functions, will be handled in LIA-2.

Page 63: OK.

Page 63-64 and 64: We believe that this shold be covered by a more general discussion of the relationship between the binding standard and the implementation.

Page 75: This is really up to the C binding standard.

Page 83: No, we really mean the former Fortran standard, not the current one.

Page 97: OK.

COMMENT #9: David M. Gay

Para 2: We agree that IEC 559 bindings are needed. See C1 of [3]. The iec_559 flag will expose deviations form IEC 559 conformance.

Item 1: The modulo parameter seems to meet your needs. This suppresses overflow checking and gives a well-defined result. Suppressing overflow checking while leaving the result undefined was explicitly rejected by several previous reviewers.

Item 2: We explained our reasoning on this in <G23> of [2]. An additional function to do what you suggest is clearly permitted, just not required.

Item 3: OK.

LIAISON COMMENT: X3J4, Cobol

Hewlett-Packard: We agree that fixed point should be covered in a future part of the LIA family. However, decimal radix is covered in the current LIA-1.

Micro-Focus (add two references to Cobol): OK.

Victor Consulting: There is no Cobol annex because the Cobol standard is currently being modified to add floating point types. An example binding

seems premature at this time.

Wang Laboratories 1: LIA-1 covers multiple float radixes except for conversions. Radix conversion will be covered in LIA-2.

Wang Laboratories 2: The annexes do cover Ada, C, and Fortran, which can all support multiple integer types.

BIBLIOGRAPHY

[1] JTC1/SC22/WG11 N364R, Information technology -- Language independent arithmetic -- Part 1: Integer and floating point arithmetic, Version 4.1, 4 August 1993

[2] X3T2/93-123, Responses to U.S. Comments on ISO/IEC CD 10967-1.2 (LIA-1), September 1993

[3] X3T2/94-008, Proposed U.S. Position on ISO/IEC DIS 10967-1 (LIA-1), 1 March 1994 (attached below)

PROPOSED U.S. POSITION ON ISO/IEC DIS 10967-1 (LIA-1)

4 March 1994

The U.S. votes to APPROVE progression of ISO/IEC DIS 10967-1 (LIA-1). In addition, the following comments are submitted.

A1:
page vi, paragraph 6
Delete it. In fact it does not comply with US comment 1.1 to 2nd CD LIA-1.

A2:
page vi, paragraph 7
Delete the last sentence. It does not comply with US comment 1.1 to 2nd CD LIA-1.

A3:
page vi, last paragraph
Delete it.

A4:
page vii, second line
Replace "characterize" by "describe some of".

A5:
page vii, first paragraph
Second and third aim need to be better formulated.

A6:
page vii, 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.

A7:
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."

A8:
page 2, g)
Delete it. It contradicts with requirements when iec_559 is true.

A9:
page 2, last sentence
Delete it.

A10:
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.

A11:
page 5, "error", (2)
Should make it clear that except for those phrases, error and exception are not synonyms.

A12:

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

A13:
page 5, "exceptional value"
See comments A10 and A11 about continuation value and exception.

A14:
page 5, "notification", second line
Replace "results in a notification" by "causes a notification".

A15:
page 5, "rounding"
Needs a meaningful definition.

A16:
page 6, "round to nearest"
Add text along these lines: "If the adjacent values are equidistant from u, either may be chosen according to precise rules which will be properly documented, and available at run time".

A17:
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.

A18:
page 7, first paragraph
Change it to "Whenever an arithmetic operation causes an exception, ...".

A19:
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.

A20:
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.

A21:
page 12, section 5.2.2
This is at best misleading. Section 5.2.9 will define iec_559. In case iec_559 is set, these operations are required to have "larger" signatures, 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.

A22:
sections 5.2.4, 5.2.5, 5.2.6, 5.2.7
US comment 5.4 to 2nd CD LIA-1 referred to these and it was not implemented. The US acknowleadges 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 add* 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.

A23:
sections 5.2.4, 5.2.5, 5.2.6, 5.2.7
There is no point in going through detailed editing comments since comment A22 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 rnd rnd, 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 5.2.4 talks of requirements in connection with add* which is not required, the last sentence of 5.2.5 is grammatically bad.

A24:
page 17, section 5.2.8, Note
The reference is probably to A.5.2.8, not A.5.2.5.

A25:
page 17, fifth line from below
This is factually incorrect. The behavior is specified when iec_559 is set.

A26:
page 17, last three lines
The definition of rnd_style needs to also accommodate the case when iec_559 is set.

A27:
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.

A28:
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.

A29:
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 is to be done a major part of this standard becomes useless.

A30:

section 7
Section 7 is still too vague. It needs to be improved.

B1:
The current definition of integer negate is

```
neg_I(x)        = -x                  if -x in I
                = integer_overflow    if -x not in I
```

The last line (integer_overflow) should be changed to

```
      = wrap_I(-x)          if -x not in I and modulo = true
      = integer_overflow    if -x not in I and modulo = false
```

B2:
The current definition of integer absolute value is

```
abs_I(x)        = |x|                 if |x| in I
                = integer_overflow    if |x| not in I
```

The last line (integer_overflow) should be changed to

```
      = wrap_I(|x|)         if |x| not in I and modulo = true
      = integer_overflow    if |x| not in I and modulo = false
```

B3:
The current definition of integer to integer conversion is

```
cvt_{Ia->Ib}(x) = x                 if x in Ib
                = integer_overflow  if x not in Ib
```

The last line (integer_overflow) should be changed to

```
      = wrap_Ib(x)          if x not in Ib and modulo_Ib = true
      = integer_overflow    if x not in Ib and modulo_Ib = false
```

where modulo_Ib is the modulo parameter for type Ib.

B4:
The current definition of floating point to integer conversion is

```
cvt_{F->I}(x)   = rnd_{F->I}(x)       if rnd_{F->I}(x) in I
                = integer_overflow    if rnd_{F->I}(x) not in I
```

The last line (integer_overflow) should be changed to

```
      = wrap_I(rnd_{F->I}(x))  if rnd_{F->I}(x) not in I and modulo_I = true
      = integer_overflow       if rnd_{F->I}(x) not in I and modulo_I = false
```

where modulo_I is the modulo parameter for type I.

B5:
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 75.

```
(unsigned int) x,      (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.

B6:
The code example in annex G.3 is not legal ADA (it uses = rather than :=).
Replace it by

```
Approx, Previous_Approx: Float;
N: constant Float := 6.0;              -- an arbitrary constant value

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;
```

B7:
WG11 should consider adding the following clause to the Rationale.

A.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 J 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->I}(x) + cvt_{J->I}(y) = cvt_{J->I}(x)
```

in I.

B8:
In annex C, clarify that a binding standard for LIA-1 *should* include bindings for all optional IEC 559 features, even though not all implementations of IEC 559 will provide those features.

B9:
John Reid of Rutherford Labs has suggested various improvements to the example Fortran binding. John Klensin of MIT has done the same for PL/I. WG11 should consider consulting with these experts to produce improved versions of the example Fortran and PL/I binding.

C1:
The US feels that binding standards for LIA-1 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.

Resolution of International Comments on LIA-1

May 1994

There were 18 votes to approve LIA-1 and 2 votes to disapprove (France and Sweden). The French no vote can be reversed by correcting the French translation of the document's title. It appears that the Swedish no vote cannot be resolved without substantial normative changes. These would set the project back to the DIS or even CD stage, and would risk the loss of one or more of the current yes votes.

Comments from France

Change French title [ACCEPT]:

WG11 agrees that the current French title of LIA-1 is incorrect, and urges the ITTF to adopt the title submitted by AFNOR.

Page 58 [ACCEPT]:

We will delete this comment on Fortran.

Page 83 [ACCEPT IN PART]:

This remains a correct comment about the (now obsolete) Fortran 77 standard. We will place it in the past tense.

Page 91 [ACCEPT IN PART]:

The parentheses will be removed from the print statement. The expression involving "HUGE" seems no less portable (in a formal sense) than any other expression containing a numeric literal.

Page 92 (G3) [ACCEPT]:

The assignment symbol will be corrected.

Page 92 (G4) [ACCEPT]:

The call keyword will be added.

Annex H [ACCEPT]:

The ANSI reference will be corrected.

Additional Comments from France

Three of the five additional comments are repetitions of comments addressed above. For the other two:

Page 82 [REJECT]:

Fortran's EXPONENT function is not quite the same as LIA-1's EXPON.

Page 93 [ACCEPT]:

X will be replaced by Y as appropriate.

Comments from Sweden

The recommended changes are all substantial normative changes. Adopting them would require reballoting LIA-1 at the DIS or even CD level. Worse, adopting these recommendations would probably change at least one national vote from APPROVE to DISAPPROVE, resulting in no net improvement.

1) Add natural numbers [REJECT]:

Unbounded unsigned integers are not currently included in LIA-1 because no standard language provides such a type, and because it would require yet another parameter. However, WG11 would like to invite Sweden to propose an addendum to LIA-1 on this topic.

2) Remove modulo integers [REJECT]:

Modulo integers do have legitimate (if limited) uses. They are an important datatype in C. We have received many expert comments stating that including a definition of modulo integers is essential. (Note that languages and implementations are not required by LIA-1 to provide modulo integers.)

Additional text will be added in the rationale which describes the hazards of modulo integers, and recommends that whenever systems provide modulo integers, that they provide a corresponding non-modulo integer type as well.

3) Replace add* by + [REJECT]:

Add* is needed to describe at least one prominent floating point architecture. If the industry evolves as expected, it may be appropriate to remove add* at the first 5-year review.

4) Div/Rem/Mod [ACCEPT IN PART]:

The truncation variants of div and rem are needed for compatibility with Fortran. Removing them will not change Fortran (or any other language), but will make it just that much harder to document a language's choices.

A similar argument applies for the Pascal variant of mod.

However, additional text will be added to clause A.5.1.3 to explain the disadvantages of the truncating div and rem, and to recommend the mathematically superior versions.

The superscripts on div, rem, and mod will be changed as follows:

div^t and rem^t will be used for truncating div/rem
div^f and rem^f will be used for flooring div/rem
mod^a will be used for mod of any modulus
mod^p will be used for mod restricted to positive modulus

A note will be added mentioning that rem^f is equivalent to mod^a.

Annex E will be reviewed for any inconsistencies with language standards.

5) Convert F to I [REJECT]:

Several standard languages choose to leave this conversion weakly defined (just as LIA-1 does). The three specifically recommended

conversions (ceiling, floor, and nearest) are already in the first draft of LIA-2.

Comments from the United Kingdom

Clause 4.1 [ACCEPT]:

The word "non-empty" will be added.

Comments from the United States

A1 [REJECT]:

No rationale for this change was given, and no alternative text was submitted.

A2 [REJECT]:

No rationale for this change was given, and no alternative text was submitted.

A3 [ACCEPT IN PART]:

The final sentence will be rewritten as "... particular set of parameter values is selected, and all required documentation is supplied, the resulting information should be precise enough to permit careful numerical analysis."

A4 [ACCEPT]:

"Characterize" will be replaced by "describe".

A5 [REJECT]:

No explanation of the problem was given, and no alternative text was submitted.

A6 [REJECT]:

The benefits section was reexamined. No false promises were found.

A7 [ACCEPT]:

The phrase beginning "to determine" will be replaced by "to the programmer".

A8 [ACCEPT IN PART]:

The following clarifying sentence will be added: "However, specifications for such values are given in IEC 559."

A9 [REJECT]:

No rationale for this change was given, and no alternative text was submitted.

A10 [ACCEPT IN PART]:

A note will be added to the definition of exceptional value stating that

"Exceptional values are not to be confused with the NaNs and infinities defined in IEC 559. Contrast this definition with that of continuation

value above."

A11 [ACCEPT]:

The following text will be added to the second definition of error: "Error and exception are not synonyms in any other context."

A12 [ACCEPT]:

The definition will be changed to

"Exception: The inability of an operation to return a suitable numeric result. This might arise because no such result exists mathematically, or because the mathematical result cannot be represented with sufficient accuracy."

A13 [ACCEPT]:

See resolution of A10 and A11.

A14 [REJECT]:

The existing text seems fine.

A15 [REJECT]:

No explanation of the problem was given, and no alternative text was submitted.

However, the following text will be added: "Note that a suitable representable result may not exist (see 5.2.6)."

A16 [WITHDRAWN]:

The U.S. indicated that their real concern was the completeness of information available at runtime. This is better expressed in A26.

A17 [REJECT]:

No additional changes seem to be needed.

A18 [ACCEPT IN PART]:

The text will be changed to "Whenever an arithmetic operation (as defined in this clause) returns ...". The operations in clause 5 return exceptional values in order to signify that a notification is required.

A19 [WITHDRAWN]:

The U.S. has reconsidered this recommendation.

A20 [ACCEPT]:

The phrase "unbounded extension" will be used instead.

A21 [ACCEPT]:

A new definition of signature will remove this problem. The definition of signature will be changed to:

"Signature (of an operation or function): A summary of information about an operation or function. A signature includes the operation name, the minimum set of inputs to the operation, and the maximum set of outputs from the operation (including exceptional values if any). The signature

```
add_I:  I x I --> I u {integer_overflow}
```

states that the operation named add_I shall accept any pair of I values as input, and (when given such input) shall return either a single I value as its output or the exceptional value integer_overflow.

A signature for an operation or function does not forbid the operation from accepting a wider range of inputs, nor does it guarantee that every value in the output range will actually be returned for some input. An operation given inputs outside the stipulated input range may produce results outside the stipulated output range."

In addition the following note will be added to 5.2.2:

"NOTE -- Operations are permitted to accept inputs not listed above. In particular, IEC 559 requires floating point operations to accept infinities and NaNs as inputs. Such values are not in F."

A22 [REJECT]:

This issue has been addressed before. The problem is that no one can come up with suitable text. No alternative text was offered by the U.S.

A23 [ACCEPT IN PART]:

The clauses will be rewritten to replace "shall" forms with "is" forms. The word "requirements" will be avoided. The grammar of the final sentence in 5.2.5 will be improved.

A24 [ACCEPT]:

The reference will be corrected.

A25 [ACCEPT]:

We will alter the sentence to read "... is not specified by this definition."

A26 [REJECT]:

Any IEC 559 binding will include a way to find out which of the four IEC 559 rounding modes is currently selected. As for rnd_style itself, the required values should not include non-conforming modes (the directed roundings), and the two forms of nearest can be distinguished (in practice) by looking at iec_559.

A27 [REJECT]:

The U.S. did not explain why they feel this clause is incomplete, and no alternative text was submitted.

A28 [ACCEPT]:

The sentence will be changed to "different choices for rnd_F->I or nearest_F can produce different conversion operations."

A29 [ACCEPT IN PART]:

After extensive discussion, the problem seems to be that readers are not remembering the provisions of clause 2 while they are reading 6.1.1. Thus, they do not realise that an exception handling mechanism defined in a language or binding standard will take precedence over the mechanisms introduced in clause 6.

To avoid this possible misreading, the title and first three sentences of 6.1.1 will be replaced as follows:

6.1.1 Language defined notification

If the programming language in use defines an exception handling mechanism that can

a) detect the occurrence of arithmetic exceptions,
b) report such exceptions to the executing program,
c) permit the programmer to specify to compensate for such exceptions, and then
d) continue program execution,

then notifications shall be handled by that language defined mechanism.

Such a mechanism may be defined as part of the programming language standard itself or by a separate binding standard.

NOTE -- The exception handling mechanisms of Ada and PL/I are examples of language defined notification. In these languages, an exception causes a prompt alteration of control flow to execute user provided exception handling code. Other notification mechanisms, such as continued execution with special non-numeric "error values", may be appropriate for other languages.

This rephrases the current requirements in a clearer and more explicit form, but does not add anything new.

Add to the first note on p.22: "If the iec_559 parameter is true, the continuation values must be precisely those stipulated in IEC 559."

A30 [REJECT]:

The U.S. did not explain why they feel this clause is too vague, and no alternative text was submitted.

B1 [ACCEPT]:

Accepted as written.

B2 [ACCEPT]:

Accepted as written.

B3 [ACCEPT]:

Accepted as written.

B4 [ACCEPT]:

Accepted as written.

B5 [ACCEPT IN PART]:

We will add (unsigned int) and (unsigned long) to the list of C conversions on page 75. However, it became clear during discussions that the subsequent paragraph is correct as it stands.

B6 [ACCEPT]:

Accepted as written. Use boldface for "constant", "while", "abs", "loop", and "end" if practical.

B7 [REJECT]:

This is really directed at language designers. However, WG11 feels that language designers already understand this issue.

B8 [ACCEPT]:

Add the following to annex C para 2: " A programming language binding for a standard such as IEC 559 must define syntax for all required facilities, and should define syntax for all optional facilities as well. Defining syntax for optional facilities does not make those facilities required. All it does is ensure that those implementations that choose to provide an optional facility will do so using a standardized syntax."

In the next sentence, change "those facilities" to "all IEC 559 facilities." Throughout the annex, weaken "must" to "should."

B9 [ACCEPT IN PART]:

These experts (and others) should be consulted, but preference should be given to the development of normative binding standards, not polishing the examples in LIA-1.

C1 [ACCEPT]:

WG11 agrees with the U.S. on this point.

Comments from Individual Experts

A number of comments were received from individual experts. Those comments that led to changes in the text are listed here.

Page vii, last sentence (Scowen): Modify the sentence to read "the results are reliable if and only if there is no notification."

Page 7 (Schaffert): Add text to explicitly allow a language to include non-numeric values in their F, and to allow additional operations.

Page 7 (Karlsson): Use italic I when introducing maxint and minint.

Page 15 (Kulisch and Walther): Delete ";" after F*.

Page 17 (Kulisch and Walther): In the first note, replace "lower bounds" with "smaller values for rnd_error."

Page 21 (Karlsson): Change "save_indicators" to "current_indicators."

Page 23 (Kulisch and Walther): Add Fortran to the list of languages that use == for equal.

Page 24 (Gay): Change "transformation" to plural.

Page 28 (Kulisch and Walther): Clarify that K-M arithmetic is entirely compatible with LIA-1's requirements, but is stricter.

Page 28 (Kulisch and Walther): Cite the following in addition to [33]: U. Kulisch and W. L. Miranker: Computer Arithmetic in Theory and Practice, Academic Press, New York, 1981

Page 39, Annex A.5.2.0.4 (Eggert): Add to the second para: "However, be sure to read 5.2.9 and C.1."

Page 44 (Karlsson): Italicize two occurrences of x.

Page 55, first para (WG11): Change "portable" to "standard"

Page 59 (Karlsson): Delete rnd_I from middle of page.

Page 61, 4th para (Schaffert): Change "requiring full accuracy" to "requiring full conformity to LIA-1".

Page 64, Annex C.2 (Eggert): Change the spelling of divide-by-zero to divide_by_zero.

Page 68 (Karlsson): Cut "where x is an expression of type FLT"

Page 69 (Karlsson): rem^1 is misspelled as mod.

Page 69 et seq (Karlsson): When a binding for mod^1 exists, use it for rem^1 as well. (Bindings effected: Ada, Fortran, PL/I)

Page 73, Annex E.4 (Eggert): Prepend "Integer valued" to the sentence "Parameters and derived constants can be used in preprocessor expressions."

Page 74 (Jones): Alter the FLT_ROUNDS table as follows:

```
truncate    FLT_ROUNDS = 0
nearest     FLT_ROUNDS = 1
other       FLT_ROUNDS /= 0 or 1
```

Remove "and subtraction" from the subsequent note.

Page 75 (Karlsson): Replace "floor" with "intprt".

Page 76 (Jones): Use "|" for combining indicators.

Page 78 (Karlsson): Swap Lisp "rem" and "mod" as bindings for the two kinds of rem_I.

Page 84 (WG11): Restore the example binding for Pascal, appropriately updated. Check other bindings against current standards and update as needed.

Page 84 (Karlsson): Add "integer k".

Pages 84 to 87 (Klensin): Change PL/I references to
ANSI X3.74, PL/I General Purpose Subset
ISO 6522-1993, General Purpose PL/I

The editor is authorized to consult with Klensin and make appropriate modifications to the PL/I binding example to conform to the above standards.

Page 88 (WG11): Also reference IEEE 754.

Page 89, Annex F.4 (Eggert): Replace "defined in 754" with "implied by 754"

Page 90, Annex F.5 (WG11): Add a reference to the proper section in the Fortran standard. Remove the citation [1]. Add an example discussion of extended precision intermediate values.

Page 91 to 94 (Karlsson): Align closing clauses as is common in Fortran.

Page 93, G.5 and G.6 (Eggert): Explain that exception are assumed not to arise, or are being omitted for clarity.

Page 95, annex H (Kulisch): Update reference [3] to the official ISO and ANSI standards.

Page 95, ref [9] (Jones): Cut "First Edition" here and in refs. Lengthen hyphens.

Page 97, annex H (Kulisch): Add " (eds.)" after "Miranker" in reference [33].