JTC1/SC22/WG11 **N222R**

Response to International Comments on the Language Compatible Arithmetic Standard

1 March 1991

1 Introduction

The CD ballot results on the Language Compatible Arithmetic Standard [1] includes detailed comments from France and the Netherlands. Subsequently, Czechoslovakia [2] and the United Kingdom [3] submitted further comments. This document is a response to those comments. All of the comments are detailed and constructive, and we wish to thank the anonymous authors for their work.

2 Response to Comments from France

1. *The proposed document does not reflect exactly the decision of the May/June meeting of WG11...*

   The text of the LCAS was not changed, since time did not allow a full revision of the standard after the WG11 meeting. The WG11 requests have been included in version 3.0, which has just been submitted to WG11.

2. *The discussion about the NWI in the Foreword is irrelevant.*

   That discussion has been removed.

3. *Warning should be given in the Foreword that NOTES in the text are informative only.*

   ISO-CS are of the view that it is not necessary to state that NOTES are informative only. However, we have added such a sentence to the Foreword.

4. *"Denormalized" is specified in terms of non ISO standards. The full definition should be given.*

   The reference to the non ISO standard has been removed. The term "denormalized" is defined in 3.2 and the glossary, and is described more fully in 4.2.

5. *"Possible extensions to this Standard" is irrelevant and should be deleted.*

   We have deleted this subclause from the Standard, and have included a discussion of proposed future work (now assigned to WG11) in the Rationale.

6. *The term "exponent bias" in "Specifications not within the Scope" is not well known and should be defined.*

   We now define "exponent bias" in 3.2.

7. *How can a standard implementation of a language also conform to the LCAS if the language does not support all of the LCAS types?*

The LCAS requires that a signed integer type and a floating point type be supported by a conforming implementation. It is our intention that an implementation of programming language X that does not provide both a signed integer type and a floating point type can not conform to the LCAS.

8. *How can a standard implementation of a language also conform to the LCAS if the language standard supports arithmetic types other than those in the LCAS?*

LCAS places no requirement on those data types not specified in LCAS. For instance, LCAS does not specify anything about type COMPLEX as in FORTRAN 77. Similarly, unsigned integers in the C programming language do not conform to LCAS and therefore are not an integer type in the sense of LCAS. This implies that the conformity statement for an implementation is essential to understand how a language implementation conforms. This should be clearer in version 3.0 of the LCAS, which gives detailed descriptions of the connections between the requirements of the LCAS and programming languages commonly used in numeric computing, in annex B.

9. *How can a standard implementation of a language also conform to the LCAS if the language does not support all of the LCAS operations?*

To conform to the LCAS, a language implementation must provide all the operations required by the LCAS. We believe that nearly all existing language implementations, in practice, provide the ability to call to vendor-supplied libraries. However, vendors can certainly provide the user with source code that can be inserted textually in the user's own code. The LCAS does not require compilers to provide any special support for the LCAS operations. An NPL report is available giving a sample implementation of the LCAS operations in Pascal.

10. *How can a standard implementation of a language also conform to the LCAS if the language does not permit extensions, or does not provide all of the intrinsic inquiry functions required by the LCAS?*

To conform to the LCAS, a language implementation must provide all the parameters and constants required by the LCAS. We believe that nearly all existing language implementations, in practice, provide the ability to call to vendor-supplied libraries to access special constants. However, vendors can certainly provide the user with source code containing definitions for these parameters and constants that can be inserted textually in the user's own code. The LCAS does not require compilers to provide any special support, e.g. intrinsic inquiry functions, for the LCAS parameters and constants.

11. *It should be said that X is a discrete subset of R.*

We have added "discrete" to the definition of rounding function (3.2).

12. *Two's complement floating point (or other specific hardware) should not be excluded.*

Frankly, we are still undecided about two's complement. We have not changed our floating point definitions to accommodate two's complement, but we welcome input on this issue. Radix complement floating point is not uniquely defined; we know of at least two different architectures, with slightly different characteristics. There are very few radix complement floating point implementations that are current. A general set of properties for radix complement greatly complicates our definitions. We have included a discussion of these issues in the Rationale (see A.4.2.0.3).

13. *Simplify the definitions of the operations with a new "range checking" function.*

We have added a function $rnd\_and\_chk_F$ as suggested.

14. *What does "it is recommended that F_1, F_2, F_3, ... satisfy" mean?*

The sentence has been clarified.

15. *Misspelling of "shall".*

This has been corrected.

16. *"Relationship with language standards" pertains to "Conformity".*

We have provided new annexes detailing the connection between standard programming languages and the requirements of the LCAS.

17. *Conflicts between standard languages and the LCAS.*

We have removed this statement. Currently, we know of no real conflict between a language standard and the LCAS.

18. *Much of the text in "Relationship with language standards" is example.*

We have made this clearer in the text.

19. *These examples should cite the appropriate language standards.*

We have included references to the language standards.

20. *Some of the examples refer to languages not yet standardized.*

This reference occurs in an illustrative NOTE, which has no normative impact. Common Lisp is widely known, and is currently being standardized in the US. We believe it is acceptable, in informational NOTES, to reference standards work which is nearly complete.

21. *The lists of non-supported features should be gathered together.*

The list of non-supported features now appears as 1.2 "Specifications not within the scope of this standard."

22. *Page 23 was missing.*

Sorry. We will send a copy of version 3.0.

23. *The annexes should be clearly identified.*

The annexes now have the proper headings and correct references.

24. *It is stated that each primitive operation has at most one rounding error, but does "pre-rounding" introduce additional rounding errors?*

We will reword the "Approximate addition" clause to avoid misunderstanding. The cumulative effect of add^*_F followed by rnd_F is one rounding error.

25. *References to "this standard" are not always clear in the context of standard programming languages.*

We have clarified the use of "this standard".

26. *The term "ulp" is used without definition.*

We have inserted the definition of "ulp".

27. *Complete references should be given for the language standards mentioned.*

We have included citations for language standards.

28. *The standard cited is not correct.*

We have changed this to cite the Basic Encoding Rules of ASN.1. The complete reference is in the Bibliography.

29. *Change "conformity" to "A conforming system" in A.2.*

We have made this change.

30. *Misspelling of "conformity".*

We have corrected this.

31. *Include the title of ISO-9001.*

We have included the title.

32. *Reword: "if the standard were written in an informal way..."*

We have clarified this sentence.

33. *Mathematical definitions needed for set definition and manipulation.*

We have noted in 3.1 all mathematical symbols which are used in the standard. We assume a familiarity with set theory notation.

34. *In the discussion of "notification" and "exception", a harmonization should be attempted between the LCAS and the Technical Report on the preparation of programming language standards.*

The Technical Report [5] uses "exception" to mean the occurrence of a violation. The LCAS uses "notification" to mean the subsequent informing of the program or user. We will try to make this clearer.

35. *X is a discrete subset of R (A.3.2).*

We have clarified this section.

36. *Mathematical definitions needed for "implies".*

We have added the implication symbols to our list (3.1).

37. *In the definition of mod_I, need $< y$.*

We have changed "$\leq y$" to "$< y$".

38. *Define NaNs.*

We have added the definition of NaN to the text and to the Glossary.

39. *Need citations for IEEE 754.*

We have included citations for IEEE 754 and 854.

40. *Formatting of "all".*

We hope that we can avoid similar problems with LaTeX.

41. *References to specific vendor implementations should be deleted.*

The table of actual parameters from vendor systems is of substantial practical use and interest. However, we have moved it to annex F (which will be removed before final balloting).

42. *Define geometric mean.*

We have included the definition of geometric mean.

43. *Misplaced footnote.*

We have reformatted the notes to the table of machine parameters (now in annex F).

44. *Remove hyphen in "in range".*

We have made this change.

45. *Replace the word "ties".*

We have clarified these sentences.

46. *The word "model" does not seem appropriate.*

The classic Brown paper [6] on floating point uses the term "model." We are following Brown's usage.

47. *Change wording.*

We have made these changes as requested.

48. *Explain notation of half-open interval.*

We now define interval notation in A.3.2. Interval notation is not used in the standard itself.

49. *Misspelling of "ways".*

We have made the correction.

50. *Accepted comments on the body of the standard should be reflected in this annex A.*

We have incorporated many changes suggested by reviewers into both the standard and the Rationale. The list of reviewers is included in the Acknowledgements clause, in annex A.

51. *ISO references should be given for language standards that are cited.*

We include ISO citations where they exist. Some language standards are under development or have no ISO counterparts as yet, e.g. Common Lisp.

52. *Need an ISO reference for Fortran 90.*

We have included an ISO reference for Fortran 90.

53. *Citation for IEEE 754 should be found in the Bibliography.*

All citations are now found in the Bibliography annex.

54. *Definition of "axiom".*

The definition of "axiom" has been clarified.

55. *Definition of "denormalized".*

The definition of "denormalized" has been clarified.

56. *Definitions for "exception" and "violation".*

These definitions have been expanded in the Glossary and elsewhere (A.5).

3 Response to Comments from the Netherlands

3.1 Notification

The LCAS defines a notification as either the alteration of control flow such that execution can continue only because of a (user-supplied) handler, or the output of an exception report. After the notification occurs, the program execution may continue, either as a result of the handler or as the system default, using an in range value (An in range value is a floating

point number whose magnitude lies between *fmin* or *fmax* inclusive, or is an integer between *minint* and *maxint* inclusive).

The LCAS places no restrictions on the in range values that can be used, or on the violations after which execution may continue.

In Example 1, the result is the notification **overflow**. Post-notification execution is not constrained, but the in range value is not the "result" of the operation causing the notification. In particular, the LCAS does not prevent a poor choice for continuation value.

In Example 2, **overflow** is not an in range value.

The LCAS recommends but does not require that the different exceptional values be distinguishable from one another. An implementation is free to produce the same effect from all the notifications. For instance, an Ada implementation may well raise the `NUMERIC_ERROR` or `CONSTRAINT_ERROR` exception for all the LCAS notifications. In Example 3, the exceptional values may be indistinguishable.

We have clarified the text of the "Notifications" clause in the standard, and added more explanation in the Rationale.

3.2 Modulo arithmetic and the C language

As noted, LCAS does not support modulo arithmetic in the style of "C" unsigned integers. The reason for this is that such modulo arithmetic is clearly machine-dependent rather than being an approximation to the mathematical concept of integers. We have clarified our discussion of unsigned integers in the annex for the C programming language to emphasize that they are not a conforming integer type.

4 Response to Comments from Czechoslovakia

1. *It is important to clarify basic properties that are common to all programming languages. The draft could help to create standard languages.*

   We agree. However, any action to modify a language standard must be carried out within the committee responsible for that standard. Annex B gives detailed suggestions for particular language bindings.

2. *Arithmetic operations are relatively unambiguous. Similar documents dealing with topics such as assignment, indexing and parameter passing would be useful.*

   These topics are not within the scope of the LCAS. New work items could be created to cover them.

3. *Page 23 is missing.*

Sorry. We will send a copy of version 3.0.

4. *The definition of the parameter "denorm" is contradictory in its use.*

Although the IBM 370 format permits unnormalized numbers, the IBM supplied software provides no support for their use. Consequently, the observable behavior is denorm = false. Section 7.4 of IEEE standard 754 contains a definition of denormalized numbers equivalent to that in the LCAS.

5. *Why are conversion operations between floating point types of different radices omitted?*

Such conversion operations will be included in a later standard [4].

5 Response to Comments from the United Kingdom

1. *The operations shown as "optional" on pages 10 and 13 should be mandatory.*

This has been done in version 3.0.

References

[1] Summary of Voting and Comments Received on a proposal to register document N796 as a Committee Draft on: Language Compatible Arithmetic Standard. ISO/IEC JTC1/SC22 N851. October 1990.

[2] Additional comments received on document N796 – Language Compatible Arithmetic Standard. ISO/IEC JTC1/SC22 N868. November 1990.

[3] UK Member Body comment on N796 –Working Draft on Language Compatible Arithmetic Standard. ISO/IEC JTC1/SC22 N887. December 1990.

[4] Proposal to Develop a Language Compatible Mathematical Procedure Standard. ISO/IEC JTC1 N1115. December 1990.

[5] Information technology – Guidelines for the preparation of programming language standards. ISO/IEC TR 10176. 1990.

[6] W S Brown. A Simple but Realistic Model of Floating-Point Computation. ACM Trans. Math. Software Vol 7. 1981. pp445-480.