Dear WG11 Participants,

The results of the straw ballot on the proposed changes to Language Independent Arithmetic, Part 1 (LIA-1, formerly known as LCAS) suggest that a second draft for the changes is needed before proceeding to the next version of the LIA-1.

The second draft follows this introduction. The miscellaneous changes at the end of the first draft are omitted here, because no questions or objections were raised on them.

All changes from the first draft are indicated by "change bars" in the left-hand margin.

The changes to clause 2 are based on comments from Jean Bourgain, Brian Wichmann, Brian Meek, Paul Rabin, and M. Sparks:

1. Jean suggested that the language bindings in Annex B should be normative -- partly to motivate language committees to provide input.
2. Brian Wichmann feared that Jean's suggestion might offend language groups, and Paul Rabin thought the LIA-1 should not preempt debate in this way.
3. Brian Meek thought that language committees would welcome the provision of language bindings by someone else, provided they were included in the balloting process. This view was seconded by Sparks who had encountered the same problem for graphics standards.

We have included minor changes to clauses 3.2 and 4 in order to clarify the meaning of the term "exceptional value," which occurs in the signatures of arithmetic operations.

The changes to clause 5 are as follows:

1. New clause 5.1.1 is a merge of old clauses 5.1.4 and 5.1.5.
2. New clause 5.1.2 is old clause 5.1.1, in which we have made no substantive changes.

   The material on continuation values has been included in new clause 5.1.2, because this is now the only alternative for which it is relevant.
3. New clause 5.1.3 is a slightly changed version of old clause 5.1.3 (message followed by termination).
4. Old clause 5.1.2 is omitted. We have, however, mentioned it as a possible aid to debugging in annex A.5.

Mary Payne, Craig Schaffert, Jeff Dawson

To: Willem Wakker, Convenor, WG11
Ace Associated Computer Experts
Van Eeghenstraat 100
1071 GL Amsterdam
The Netherlands

From: Mary Payne, MLO1-3/B68
Digital Equipment Corp.
146 Main St.
Maynard, MA 01754
USA

PROPOSAL FOR CHANGES TO THE FIRST COMMITTEE DRAFT OF LANGUAGE INDEPENDENT ARITHMETIC, PART 1 ISO/IEC cd10967-1 Version 2 of Proposed Changes

Introduction

The editors of ISO/IEC cd10967-1 propose the changes below to the first CD. Many of these changes will dictate changes to clauses 6 and 7, as well as changes in the relevant annexes.

page 3: Revised clause 2

page 3: Clarify "exceptional value" in clauses 3.2 and 4.

pages 4 to 7: Revised clause 5

pages 8 to 11: Revised annex A.5

Revised Text for Clause 2:

It is expected that the provisions of this International Standard will be incorporated by reference and further defined in other International Standards; specifically in language standards and in language-binding standards, hereafter referred to as "binding standards". Binding standards specify the correspondence between the abstract data types and operations of this International Standard and the concrete language syntax of the language standard.

When a binding standard for a language exists, an implementation shall be said to conform to this International Standard if and only if it conforms to the binding standard. In particular, in the case of conflict between a binding standard and this International Standard, the specifications of the binding standard shall take precedence.

When no binding standard for a language exists, but a binding for that language is provided in annex B, an implementation shall be said to conform to this International Standard if and only if it conforms to the binding in annex B.

When no binding standard for a language exists, and no binding for that language is provided in annex B, an implementation conforms to this International Standard if and only if it provides at least one signed integer type, and at least one floating point type, that together satisfy all the requirements of clauses 4 through 7.

An implementation is free to provide arithmetic types that do not conform to this standard or that are beyond the scope of this standard. The implementation shall not claim conformity for such types.

An implementation is permitted to have modes of operation that do not conform to this standard. However, a conforming implementation shall specify how to select the modes of operation that ensure conformity.

NOTE - See annex C for an example of a conformity statement.

Changes to Clause 3.2 and Clause 4:

Add the following note to the definition of "exceptional value" in clause 3.2:

NOTE - Exceptional values are used as part of the defining formalism only. With respect to this International Standard, they do not represent values of any of the data types described. There is no requirement that they be represented or stored in the computing system. They are not used in subsequent arithmetic operations.

Add the same note after the last paragraph of clause 4.

5 NOTIFICATION

Notification is the process by which a user or program is informed that an arithmetic operation cannot be performed. Specifically, a notification shall occur when any arithmetic operation returns an exceptional value as defined in clause 4.

5.1 Notification Alternatives

This International Standard provides three alternatives for notification. The requirements are

a) The alternative in clause 5.1.1 shall be supplied in conjunction with any language which provides support for exception handling.

b) The alternative in clause 5.1.2 shall be supplied in the absence of language support for exception handling.

c) The alternative in clause 5.1.3 shall be supplied by all implementations.

The alternatives in clause 5.1.1 and in clause 5.1.2 both require action on the part of the application programmer in order for the notification to occur. In the event that the application programmer has not taken such action, the mechanism described in clause 5.1.3 shall be used.

5.1.1 Alteration Of Control Flow

An implementation shall provide this alternative for any language that provides a mechanism for the handling of exceptions. It is allowed (with system support) even in the absence of such a mechanism.

Notification consists of prompt alteration of the control flow of the program to execute user provided exception handling code. The manner in which the exception handling code is specified and the capabilities of such exception handling code (including whether it is possible to resume the operation which caused the notification) is the province of the language standard, not this arithmetic standard.

If no exception handling code is provided for a particular occurrence of the return of an exceptional value as defined in clause 4, that fact shall be reported to the user of that program in an unambiguous and "hard to ignore" manner. See clause 5.1.3.

5.1.2 Recording Of Indicators

An implementation shall provide this alternative for any language that does not provide a mechanism for the handling of exceptions. It is allowed (with system support) even in the presence of such a mechanism.

Notification consists of two elements: a prompt recording of the fact that an arithmetic operation returned an exceptional value, and interrogation of the recording at a subsequent time by the program or system.

The recording shall consist of four indicators, one for each of the exceptional values that may be returned by an arithmetic operation as defined in clause 4: integer_overflow, floating_overflow, floating_underflow, and undefined.

These indicators shall be clear at the start of the program. They are set when any arithmetic operation returns an exceptional value as defined in clause 4. Once set, an indicator shall be cleared only by explicit action of the program. The implementation shall not allow a program to complete successfully with an indicator that is set. Unsuccessful completion of a program shall be reported to the user of that program in an unambiguous and "hard to ignore" manner. See clause 5.1.3.

NOTE - The status flags required by IEEE 754 [1] are an example of this form of notification, PROVIDED that the program is not allowed to terminate successfully with any status flags still set.

Consider two conceptual data types E and R with the following properties:

1. The data type E is composed of the following values: integer_overflow, floating_overflow, floating_underflow, and undefined, naming the indicators in the recording,

2. A value of the data type R stores a complete recording, i.e. the aggregate state of all indicators.

The implementation shall provide an embedding of these two conceptual data types into existing programming language types. In addition the implementation shall provide the following operations to interrogate and manipulate the recordings and indicators. Let e be a value in E, and r be a value in R:

```
test_indicator(e)  = true     if indicator e is set
                   = false    if indicator e is clear

set_indicator(e)              set indicator e

clear_indicator(e)            clear indicator e
```

save_recording() = r where r is the current recording

restore_recording(r) replace the current recording with r

When any arithmetic operation returns an exceptional value as defined in clause 4, after recording the event, an implementation shall provide a "continuation" value for the result of the failed arithmetic operation, and continue execution from that point:

- In the case of floating_underflow, the continuation value shall be rndF(exact_result) when denorm is true, or 0 when denorm is false.

- In the case of integer_overflow, floating_overflow, and undefined, the continuation value shall be implementation dependent. There are no restrictions on this continuation value. It is not required to be a valid value of the type I or F.

NOTE - The infinities and NaNs produced by an IEEE 754 system are examples of values not in F which might be used as continuation values.

NOTE - This International Standard does not specify what happens when an operation is applied to a value that is not in its input domain (as defined by the operation signature). Thus, for example, the behavior of addF on a NaN is not in the scope of this International Standard.

NOTE - No changes to the specifications of a language standard are required to implement this alternative for notification. The recordings can be implemented in system software. The operations for interrogating and manipulating the recording can be contained in a system library, and invoked as library routine calls.

5.1.3 Termination With Message

An implementation shall provide this alternative, which serves as a back-up if the programmer has not provided the necessary code for either of the other alternatives.

Notification consists of prompt delivery of a "hard-to-ignore" message, followed by termination of execution. Any such message should identify the cause of the notification and the operation responsible.

5.2 Delays In Notification

Notification may be momentarily delayed for performance reasons, but should take place as close as practical to the attempt to perform the responsible operation. When notification is delayed, it is permitted to merge notifications of different occurrences of the return of the same exceptional value into a single notification. However, it is not permissible to generate duplicate or spurious notifications.

In connection with notification, "prompt" means before the occurrence of a significant program event. For the recording of indicators in 5.1.2, a significant program event is an attempt by the program (or system) to access the indicators, or the termination of the program. For alteration of control flow described in 5.1.1, the definition of a significant event is language dependent, is likely to depend upon the scope or extent of the exception handling mechanisms, and must therefore be provided by language standards or by language binding standards.

NOTE - Roughly speaking, "prompt" should at least imply "in time to prevent an erroneous response to the exception."

5.3 User Selection Of Alternative For Notification

A conforming implementation shall provide a means for a user or program to select among the alternate notification mechanisms provided. The choice of an appropriate means, such as compiler options, is left to the implementation.

The language or binding standard should specify the notification alternative to be used in the absence of a user choice.

A.5 Notification

The essential goal of the notification process is that it should not be possible for a program to terminate with an unresolved arithmetic violation unless the user has been informed of that fact, since the results of such a program may be unreliable.

The simplest way of achieving this is to abort the program. Unfortunately, this approach conflicts with the goal of supporting a robust computing environment where the execution of a program can continue satisfactorily in all situations.

A.5.1 Notification alternatives

This standard provides a range of alternative mechanisms for notification to fit the range of implementation alternatives and requirements of both the programming language and the underlying hardware. Since this standard is concerned only with quality in numeric processing, the minimum set of notification mechanisms needed for reliable numeric results is described. This set of alternatives allows programmers to make the choice of whether to handle exceptions themselves or not, and to do so in portable, language based fashion. Programmers are also assured that when they choose not to handle their own exceptions, they will know if an exception does indeed occur across the range of conformant implementations.

Implementations are encouraged to provide additional mechanisms which would be useful for debugging. For example, pausing and dropping into a debugger, or continuing execution while writing a log file.

In order to provide the full advantage of these notification capabilities, information describing the nature of the violation should be complete and available as close in time to the occurrence of the violation as possible.

A.5.1.1 Alteration of control flow

This alternative requires the programmer to provide application specific code which decides whether the computation should continue, and if so how it should continue. This alternative places the responsibility for the decision to continue with the programmer who is presumed to have the best understanding of the needs of the application.

ADA and PL1 are examples of standard languages which include syntax which allows the user to describe this type of alteration of control flow.

Note, however, that a programmer may not have provided code for all trouble-spots in the program. In this case, recourse to program termination is probably the only viable option.

A.5.1.2 Recording of indicators

This alternative gives a programmer the primitives needed to obtain exception handling capabilities in cases where the programming language does not provide such a mechanism directly. An implementation of this alternative for notification requires no extensions to any language. The status of the indicators is maintained by the system. The operations for testing and manipulating the indicators can be implemented as a library of callable routines.

This alternative can be implemented on any system with an "interrupt" capability, and on some without such a capability.

By making use of the required status flags, this alternative can be implemented on an IEEE system. The mapping between the IEEE status flags and the LIA-1 indicators is as follows:

IEEE flag	LIA-1 indicator
overflow	floating_overflow
underflow	floating_underflow
invalid	undefined
division by zero	undefined
inexact	no LIA-1 counterpart

The LIA-1 does not include notification for inexact because non-IEEE implementations are unlikely to support inexact exceptions.

For a zero divisor, IEEE specifies an "invalid" exception if the dividend is zero, and a "division by zero" otherwise. Other architectures are not necessarily capable of making this distinction. In order to provide a reasonable mapping for an exception associated with a zero divisor, the LIA-1 specifies undefined, regardless of the value of the dividend.

An implementation must check the exception recording before successfully terminating the program. Merely setting a status flag is not regarded as adequate notification, since this action is too easily ignored by the user and could thus damage the integrity of a program by leaving the user unaware that an unresolved arithmetic violation occurred. Hence this

International Standard prohibits successful completion of a program if any status flag is set. Implementations can provide system software to test all status flags just prior to completion, and if any flag is set, provide a message before termination.

The mechanism of recording indicators proposed here is general enough to be applied to a broad range of phenomena by simply extending the value set E to include indicators for other types of conditions. However, in order to maintain portability across implementations, such extensions should be made in compliance with other standards, such as language standards.

A.5.1.3 Termination with message

This alternative supports the conservative view that the only reasonable action following a notification is termination of the program since (in this view) all violations are the result of programming errors. This response certainly fits the criterion of making the notification "hard to ignore". It can be supported with relatively little effort by all implementations.

This "lowest common denominator" response is regarded by many as unnecessarily harsh and it certainly conflicts with the goal of robust execution. However, it serves as the only viable action if the programmer has not provided the necessary code for the other alternatives.

A.5.2 Delay of notification

Many modern floating point implementations are pipelined, or otherwise execute instructions in parallel. This can lead to an apparent delay in reporting violations, since an overflow in a multiply operation might be detected after a subsequent, but faster, add operation completes. The provisions for delayed notification are designed to accommodate these implementations.

Parallel implementations may also not be able to distinguish a single overflow from two "almost simultaneous" overflows. Hence, some merging of notifications is permitted.

Imprecise interrupts (where the offending instruction cannot be identified) can be accommodated as notification delays. Such interrupts may also result in not being able to report the kind of violation that occurred, or to report the order in which two or more violations occurred.

In general the longer the notification is delayed the greater the risk to the continued execution of the program.

A.5.3 User selection of alternative for notification

On some machine architectures, the notification alternative selected may influence code generation. In particular, the optimal code that can be generated for alternative 5.1.1 may differ substantially from the optimal code for alternative 5.1.2. Because of this, it is unwise for a language or binding standard to require the ability to switch between notification alternatives during execution. Compile time selection should be sufficient.

If a system had a mode of operation in which errors were totally ignored, then for this mode, the system would not conform to this International Standard. However, modes of operation that ignore errors may have some uses, particularly if they are otherwise LIA-1 conformant. For example, a user may find it desirable to verify and debug a program's behavior in a fully LIA-1 conformant mode (error checking on), and then run the resulting "trusted" program with error checking off. Another non-conformant mode could be one in which the final check on the notification indicators was suppressed.

In any case, it is essential for an implementation to provide documentation on how to select the various LIA-1 conforming notification alternatives provided.