Dear WG11 Participants,

As promised to WG11 at the Vienna meeting, the proposed changes for the LCAS are enclosed.

The rewrite proposed for Clause 2 is in response to the AFNOR comment which accompanied its negative vote to progress 10967 from CD to DIS.

The complete rewrite of Clause 5 and Annex A.5 is in response to a number of US comments to X3T2. These comments complain that the LCAS does not allow the default "non-stop" mode of execution specified in IEEE 754 (IEC 559).

We have a few issues on which we would like guidance:

1. We (the editors) consider that Clause 5.1.2 (provide a message and continue execution) does not serve a useful purpose. It is the only alternative that permits a program to continue after an error without any knowledge (available to the program) that an error has occurred. It is included because it is part of the "package" agreed to at the Vienna meeting. We propose that it be removed.
2. The UK wants 5.1.3 to be the default.

   The IEEE 754 people want 5.1.1 to be the default.

   How should WG11 reply to these concerns?

   In the absence of a default requirement by a standard language or language binding standard, the present rewrite leaves the default implementation dependent.
3. Are the specifications in Clauses 5.1.4 and 5.1.5 adequate for and compatible with standards for languages such as Ada, PL/I and C which provide for user exception handling?
4. We (the editors) propose that alternatives 5.1.4 and 5.1.5 be merged into a single clause titled "Alteration of Control Flow," which allows continuation of execution either by invocation of a program exception handler, or by a "jump" mechanism. This should make the subsequent text easier to read.
5. After answering the above questions, please indicate your preferences on the following questions:

   If a language standard for the language in use stipulates an exception handling mechanism, clearly the implementation must provide that mechanism. Which of the five LCAS alternatives (if any) are REQUIRED to be provided by an LCAS conformant implementation IN ADDITION to the language specified one? Which ones are PERMITTED to be provided?

   IF a language standard for the language in use DOES NOT stipulate an exception handling mechanism, which of the five LCAS alternatives are REQUIRED to be provided by an LCAS conformant implementation? Which ones are PERMITTED to be provided?

Mary Payne

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 ISO/IEC 10967:1991, JTC1/SC22/WG11 N229

Introduction

The editors of ISO/IEC 10967 (LCAS) propose the changes below to the 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

pages 4 to 7: Revised Clause 5

pages 8 to 10: Revised Annex A.5

pages 11 to 12: Eight relatively minor changes

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. Binding standards may require conformity to all, or only some, provisions of this International Standard; and they may override some of the provisions of this International Standard.

When a binding standard exists, an implementation for that language shall be said to conform to this International Standard if and only if it conforms to the binding standard.

When no binding standard exists, an implementation for that language shall be said to conform to this International Standard if it provides at least one integer and/or one floating point data type and satisfies all of the requirements of Clauses 4 through 7 for the data types provided.

> NOTE - In the absence of a binding standard, a conforming implementation should follow the appropriate language binding found in Annex B of this International Standard.

Revised text for Clause 5:

5 NOTIFICATION

Notification is the process by which a user is informed that an arithmetic operation cannot be performed. Specifically, a notification shall result when any arithmetic operation returns an exceptional value, or when resource exhaustion occurs.

5.1 Notification Alternatives

An implementation shall provide all of the alternatives for notification specified by clauses 5.1.1, 5.1.2 and 5.1.3. In addition, the alternatives specified by clauses 5.1.4 and/or 5.1.5 shall be provided for languages whose standards support those notification mechanisms.

5.1.1 Recording Of Indicators

Notification consists of making a recording that can be interrogated at a subsequent time by the program or system. The recording shall consist of four indicators, one for each of the exceptional values: integer_overflow, floating_overflow, floating_underflow, and undefined. These indicators shall be cleared at the start of the program. 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.

> 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.

The implementation shall provide:

1. A data type E composed of the following values: integer_overflow, floating_overflow, floating_underflow, and undefined, naming the indicators in the recording,
2. A data type R to store a complete recording, i.e. the aggregate state of all indicators, and
3. The following operations to interrogate and manipulate the recording, where e is a value in E, and r is 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

5.1.2 Continuation With Message

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

5.1.3 Termination With Message

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.

5.1.4 Alteration Of Control Flow

Notification consists of prompt alteration of the control flow of the program such that execution shall continue only as a result of explicit action specified in the program.

5.1.5 Execution Of Program Defined Handler

Notification consists of prompt execution of a program defined exception handler, followed by continuation of execution.

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 multiple notifications into a single notification. However, it is not permissible to generate duplicate or spurious notifications.

The recording of indicators shall not be delayed beyond a point where the program (or system) attempts to access the indicators as described in 5.1.1; or beyond the scope of user provided exception handling code as described in 5.1.4 or 5.1.5.

The precise definition of "prompt" is language dependent, 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 alteration of control flow."

5.3 Continuation Values

When execution continues, as provided in clauses 5.1.1, 5.1.2, 5.1.4 and 5.1.5, it is usually necessary for the implementation to supply a "continuation value" for use as the returned value of the arithmetic operation involved.

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 such 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.

5.4 Default Notification Mechanism

If a default among the alternative notification mechanisms is specified by the language or binding standard, then that alternative shall be the default for an implementation conforming to this International Standard.

In the absence of a language or binding specification for a default, the default shall be implementation dependent.

5.5 User Selection Of Alternative For Notification

A conforming implementation shall provide a means for selecting among the alternate notification mechanisms supported. The choice of an appropriate means, such as compiler options, is left to the implementation.

5.6 Resource Exhaustion

Because resource exhaustion is by nature implementation dependent, the means of notification is likely to be implementation dependent as well. Resource exhaustion shall be reported by one of the alternatives provided in Clause 5.1. No means for selection among the alternatives is required.

Revised Text for Annex A5

A.5 Notification

The essential goal of the notification process is that it should not be possible for a program to terminate without the user being aware that an unresolved arithmetic violation has occurred, and hence that the results 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.

A.5.1.1 Recording of indicators

This alternative gives a programmer the primitives needed to obtain exception handling capabilities in cases where the programming language does not have syntax to describe the capabilities directly. In order to take full advantage of these 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.

The IEEE status flags provide an implementation of this alternative for notification. The mapping between the IEEE status flags and the LCAS indicators is as follows:

IEEE flag	LCAS indicator
overflow	floating_overflow
underflow	floating_underflow
invalid	undefined
division by zero	undefined
inexact	no LCAS counterpart

Non-IEEE implementations are unlikely to support inexact exceptions, which are therefore not portable.

For a zero divisor, IEEE specifies a division by zero exception if the dividend is not zero, and an invalid exception if the dividend is zero. In order to provide a reasonable mapping for an exception associated with a zero divisor, the LCAS specifies undefined, regardless of the value of the dividend.

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. It also seems a reasonable course of action when the user has not made any other provisions. However, this "lowest common denominator" response is regarded by many as unnecessarily harsh and it certainly conflicts with the goal of robust execution.

A.5.1.4 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. In order to make this possible, the programming language must have syntax which allows the programmer to describe conditional branches in control flow. Note, however, that a programmer may fail to provide code for all trouble-spots in the program. In this case, recourse to program termination is probably the only viable option.

A.5.1.5 Execution of program defined handler

This alternative has the same advantages as the alteration of control flow described in A.5.1.4, but for languages which have syntax for user defined exception handlers.

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 might be detected after a subsequent, but faster, add 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.5.5 User selection of alternative for notification

If a system had a mode of operation in which errors were totally ignored, then for this mode, the system would not conform. However, modes of operation that ignore errors may have some uses, particularly if they are otherwise LCAS conformant. For example, a user may find it desirable to verify and debug a program's behavior in a fully LCAS conformant mode (error checking on), and then run the resulting "trusted" program with error checking off.

In any case, it is essential for an implementationto provide documentation on how to select the various LCAS conforming notification alternatives provided.

Miscellaneous Smaller Changes.

These miscellaneous changes are presented in outline form:

1. Clause 4.2.4, Range Checking.

   The range checking logic will be modified to reflect the changes in the notification procedure. These changes will not be substantive; they will be made for consistency with the changes in Clause 5. Details will be given, once Clause 5 has stabilized.

2. Clause 4.2.3, Rounding. Add the condition that

   rndF(-x) = -rndF(x)

   Remove mention of other rounding rules.

   Require an implementation to supply a parameter giving the error bound of rndF as a multiple of ulps in the returned result. The name of this parameter is yet to be specified.

   List the additional identities satisfied by addF, subF, and mulF

   addF(-x,-y) = -addF(x,y)
   subf(-x,-y) = -subF(x,y)
   mulf(-x,y) = mulf(x,-y) = -mulF(x,y)

   in Annex A.4.2.9.

3. Clause 4.3, Conversions.

   A rnd-nearest rounding function satisfies

   |rnd-nearest(x) - x| <= (r^(e(x) - p))/2

   in addition to satisfying the constraints on a rounding function specified in Clause 4.2.3.

   Conversions from integer to floating point and from one floating point type to another shall use a rnd-nearest rounding function.

   NOTE - This definition of rnd-nearest allows either direction of rounding for the "half-way" case.

4. Clause 4, The Arithmetic Types.

   Split the exceptional value overflow into two values integer-overflow and floating-overflow.

5. Clause 4, The arithmetic types.

   Remove the exceptional value zero-divisor, and replace it by undefined in Clauses 4.1.1, 4.1.3, 4.2.2, and 4.2.6. This change is needed for compatibility with the treatment of the division by zero exception of IEEE 754.

6. Clauses 4.1.1, Integer operations and 4.1.3, Integer axioms.

   Add signI(x) to the list of integer operations (requested by UK)

   signature: signI: I --> I

   axioms: signI(x) = -1, 0, +1 for x negative, zero and strictly positive, respectively.

7. Clause 4.2.6, Axioms.

   Change the signF(x) axioms to signF(x) = -1, 0, and +1 for x negative, zero, and strictly positive, respectively.

8. Clauses 4.2.2 and 4.2.6 and Annex 4.2.2

   Remove sqrtF from the LCAS and include it instead in the Language Compatible Mathematical Procedure Standard.