To: X3J3
From: /hpc
Subject: Interval Intrinsics (exceptions)

For every intrinsic function, we need to have an operational definition of the domain, and a standard way to provide a standard conforming result. Historically, we have left this to the realm of *general understanding*. So, for example, SQRT(-1.) is left to a *mathematical definition* which is either an error (non-standard conforming) or on IEEE processors it is defined in the IEEE sense. Similarly we need to know what should SQRT(x) return when x is an interval [-1.,1.]?

After enough subgroup time (in between meetings, and at this meeting) it has been agreed that what is needed is an *IEEE TR-like* approach. There shall be a way for a user to inform the processor that this should be treated as a runtime error, or to continue. That the user shall have a way to discover if such an error occurred. Such discovery is often called *retrospective*.

Discussion:

For cases such as SQRT([-1,1]), one may wish to think of there being a *true* answer in the middle, that is 0, with an uncertainty of 1. SQRT(0) is well defined, and is 0. Similarly for any *true* value greater than 0. *Ex post facto* we can consider the negative values as having been overestimates of the lower bound (that is broader than was necessary, or even useful). So the function should compute as if the interval were SQRT([0,1]). There are many situations where this is the appropriate result and there is a camp which feels that this is the appropriate definition.

Of course, there is another way of handling the same situation. The *true value* could actually be -.00001, and therefore the appropriate result should be an error condition. There are situations where this is precisely what a sophisticated user would want (or a naïve one for that matter :>).

There is a difference in category when the entire interval is outside the definitional domain. SQRT([-2.,-1]). Everyone can agree that this is an error, in the same sense that SQRT(-1) is an error.

Thus there are two kinds of interval "exceptions". For want of better terms these may be called *soft* and *hard* exceptions.

Proposal Outline:

Interval exceptions, along the lines of IEEE exceptions are defined. Think of these interval status flags as parallel to the IEEE exceptions. But where there is one bit for IEEE there are two bits for intervals. This is not a complete proposal, nor should it be inferred that **all** IEEE flags are intended to be replicated. Nor that we need to provide quite the same level of flexibility. Our current thinking is that all interval processors shall be required to support these flags and to support halting. Unlike the IEEE situation, there is neither existing practice nor compelling hardware arguments to justify the extra complexity for users to complicate their codes with tons of conditional logic.

Exception flags are sticky flags, whose value is initially clear. A processor shall provide a halting mode, which is imprecise. The initial halting mode is halting (both soft and hard). Users must employ INTERVAL_SET_HALTING_MODE to establish nonstop behavior. Halting shall be controlled separately for hard and soft errors. The exception flags may be cleared by the user, stored by the user, etc. paralleling the IEEE_GET and SET elemental subroutines.