IEEE Task P854

Minutes, 20 April 1983

The radix-free floating-point working group of the Microprocessor Standards subcommittee of the IEEE Computer Society met from 9:00 a.m. to 4:45 p.m. at the Argonne National Laboratories, Argonne, Illinois. Seven people were in attendance.

Minutes from the 8 March meeting in Berkeley were agreed to.

The next meeting of P854 will be on Tuesday, June 7, 1983 in Denver in conjunction with the SIAM National Meeting. It will be preceeded in the day by a mini-symposium on the status (technical and procedural) of P754 and P854 comprising talks by Stevenson, Cody, Hough (vice Coonen), and Kahan.

John Palmer's proposal for a session at WesCon in October has been accepted. He will make firm tentative commitments from Hough, Taylor, Owen, and Kahan.

On the continuing question of publication policy, Ris and Cody had paid a visit to Dave Jacobsohn at Argonne (formerly an officer in the IEEE Computer Society) for advice. His advice was that works such as P754 Draft 10.0 should be copyrighted by somebody, it might as well be IEEE, and that as long as the cost of "official" copies was modest, there should be no problem with the IEEE making a charge provided that there be no restrictions on further copying not for sale. This was presented to the meeting and received unanimous endorsement as a sensible position. Cody will continue his attempts to get the IEEE position clarified and offer the above formulation as a suggestion wherever confusion is encountered in the IEEE hierarchy.

<u>Changes to Draft 0.8.</u> Changes made at the previous meeting to draft 0.7 were reviewed. The only changes made to draft 0.8 were in section 5.6.

At the end of the second paragraph was left hanging the question of when inexact is signalled on the conversion of very long decimal strings. Up to the previous meeting, the view was that inexact must be signalled precisely in all cases in decimal implementations and could be fuzzed some to give misleading signals in binary. That view under close scrutiny was found likely to be unrealistic in some practical situations, but Ris had some reservations about not requiring an inexact signal when a decimal string only one digit longer than a basic precision could otherwise provoke a .9 ulp error without exception. The observation that P754 already permits as much as .8796 ulp in double and the generalization formulae permit as much as .9671 ulp in a p=93 MaxD=29 implementation flattened that objection, and accordingly the formulation adopted becomes quite simple. That paragraph will now read, "...to other decimal digits, typically 0, and should signal inexact when nonzero digits are so discarded." (The words following the comma are new.)

In the fifth paragraph, the last sentence is to end after "...namely, MaxD digits." The footnote is also to be transferred. (The remainder of the sentence was an implementation hint.)

In Table 2, the expression for MaxD in the case b=2 was simplified by noting that the minimum of the two values presented would always be the first and therefore the expression could be simplified. For the expression for MaxN it was agreed that the appropriate form is the floor of some integer times log 5Õ(2), but the question of how to specify the integer is now the sticky point. Various plausible alternatives are:

p + ceil (log 2Õ (E maxÕ - E minÕ)) (1)

p eÕ (2)

p + 7 (3)

p (4)

where p eÕ is intended to mean the minimal extended precision induced by the basic precision (max (1.2p, p+7)). There was agreement that (4) is too weak and that there is little to choose between (2) and (3). Kahan preferred (1) on grounds of function, Cody and Ris preferred (2) on grounds of economy of specification and fewest additional implementation considerations, others were unsure. For the time being the specification will be the minimum of (1) and (2) until stronger arguments appear. Meanwhile, the infamous p cÕ has disappeared.

Language Issues. Kahan opened the discussion with the assertion that P754 explicitly intends that all exceptions be treated as if run-time exceptions (although that intention must be inferred from the draft). This leads to recurring language questions with potentially unpleasant answers.

The area which exposes most of the possible things which can go wrong is constant propagation (or "folding") at compile-time. What happens when a robust code needs to generate infinity? Should something like "X = 1.0/0.0" be written? What happens when a robust code want to generate zerodivide? How can we separate the two? How can we even compute something like "1.0/3.0" when the rounding mode in effect at execution may not be knowable at compile time?

These questions have been faced by compiler writers in the past and have been handled sometimes more gracefully, sometimes less. But the problems are rather tame until the inexact exception is considered. And with inexact, one doesn't need to look to constant folding as a source of trouble. If the inner loop of a computation contains the constant 0.3 in a binary implementation, do we expect an inexact signal to be generated each time through the loop at precisely the time the semantics of the language would have called for the conversion?

Compiler diagnostics for exceptions generated at compile time? Directives to the compiler at the level of compilation units (e.g., Fortran subroutines)? At finer levels? (Now we get involved with the language itself.) Compile time flags that remain sticky up to the start of execution? No matter what is done the user can be misled under some circumstances. When this is the state of affairs, is pessimism preferable? (Cf. discussion above about how much can be thrown away on conversion without notice.) Compile-time flags which can be set, tested, and precipitate conditional compilation?

In the long run we hope the flags will have diminished importance as we become comfortable with the calculus of infinities and NaNs. In the short run, flags are predominantly for the benefit of those who are not consciously using the new facilities.

No conclusions were reached and there is not a solid feeling that all the issues have yet been identified.

Fred Ris