IEEE Task P854

Minutes, 20 October 1982

(These minutes were circulated in a handwritten draft at the December 3 meeting where substantive changes were recorded. This version reflects those changes.)

The radix-free floating-point working group of the Microprocessor Standards subcommittee of the IEEE Computer Society met from 10:00 a.m. to 1:45 p.m. at the IBM Palo Alto Scientific Center. Eleven people were in attendance.

Minutes from the August meeting were agreed to.

The next meeting of P854 will be Friday, December 3 at Apple in Cupertino.

Relationship of Basic and Extended Precisions. Consideration returned yet again to the specification of how much more precision must be carried above a nominal "basic" format to qualify as "extended". The dominant considerations are:

1. Efficiency. If requirements get too close to double length, it isn't worth it.

2. Facilitation of accurate binary <-> decimal conversion in the "inner range" (cf. August minutes).

3. Guard digits for inner product and other extended sum accumulation.

4. Guard digits to recover x**y accurately from y * ln (x).

For binary, consideration (2) was shown to be satisfied at the August meeting provided seven extra bits were maintained. Consideration (1) is not adversely impacted; hence, it was decided to require a minimum of seven extra bits in binary. Consideration (2) is irrelevant in decimal.

The number of guard digits desirable for accumulation depends to first order on the number of items to accumulate. (We cannot address the zeroth order dependence on the data themselves.) Vector lengths in turn depend on an implementation's address space which in turn is often related to word size. It would seem in many cases that two or three digits in decimal offers the best mix of considerations (1) and (3). But how to decide?

Consideration of specific plausible implementations led to the empirical discovery that a minimum 20% increase in number of digits seems to lead to palatable consequences; hence, this condition was decided to be made mandatory subject to the additional constraint above in binary.

In consequence, for binary we have

basic precision	increment for extended	
	18-35	≥ 7
	36-40	≥ 8
	41-45	≥ 9
	46-50	≥10
	51-55	≥11
P754	24	
P754	53	

For decimal, we have

basic precision	increment for extended	
	6-10	≥ 2
	11-15	≥ 3
	16-20	≥ 4

Although the 20% rule cannot be derived from first principles, its expression is so economical, and its consequences are so agreeable that it is made mandatory.

Considerations (1) and (4) have been previously shown to be antagonists in many plausible decimal implementations and additionally, not exactly satisfied in P754. Hence, the requirement will be phrased as a "should" but will remain in the text of the standard to emphasize the importance of a consideration which is not immediately apparent to a potential implementer.

There was some discussion about minimal acceptable precision. If 6-digit decimal is allowed, then so must be 18-bit binary. That being so, why not relax to 15-bit or 16-bit binary because of their ubiquitous nature? Answer: 6-digit decimal in a 32-bit representation is already a concession to help motivate something which is difficult today without custom circuit designs--it is certainly not desirable. Compromising the minimally required precision further could severely impact software portability. Hence, no change.

Binary <-> Decimal Conversion. The magic constant 0.47 which appears in P754 has been shown for the specific circumstances of P754's precision and exponent ranges to represent a reasonable compromise among functionality, efficiency, and testability. It cannot be reduced by much without impacting efficiency, and if made closer to 0.5 makes the job of testing an implementation considerably more tedious. While we cannot adduce a first principle argument for a specific value, we recognize the importance of the form in which the constant 0.47 appears. Hence, we rename the magic number epsilon and require the implementation to fix some value for itself strictly less than 0.5, while observing that the value 0.47 appearing in P754 has been found to be economically achievable.

Conversions of NaNs and Infinities. Conversion of a NaN to external form shall produce a string of the form:

+/-“ NaN ....“

meaning an optional sign, the string "NaN" in optional mixture of upper/lower case, and optional further identification of the NaN. All options and the interpretation of identification as may be supplied is implementation-defined.

On input, a system which has produced an identifiable NaN as output and is given that string on input may choose to recognize it in a system-defined way. In all other cases, including NaNs not recognized from foreign systems or gibberish generally, invalid operation shall be signalled. A token which is recognized as legal output but for which specific information is not handled specially, may be converted to a signalling NaN without exception. This shall be the only way that a signalling NaN can be created by operations in the standard.

We failed to establish what appropriate strings to represent infinities should be. We also failed to establish a mechanism to decide. Plausible candidates are signs followed by "Inf" or "Infinity" or "1/0". (The latter notation further suggests "0/0" as an alternative for "NaN".)

Another open issue is how to treat a zero-valued external string. It is suggested (for future consideration) that in converting a zero-valued floating-point number to a Fortran-F-format-like string that simply the sign and a single digit zero be produced, without radix point and trailing zeroes as would occur when a small but non-zero value were rounded to fit such a fixed-point format.

Fred Ris

/lar