Why do we need a floating-point arithmetic standard?

W. Kahan

University of California at Berkeley
March 5, 1981

Contents

Why do we need a floating-point arithmetic standard?

W. Kahan

University of California at Berkeley

~~February 17~~, 1981

Mar. 5,

> "...the programmer must be able to state which properties he requires... Usually programmers don't do so because, for lack of tradition as to what properties can be taken for granted, this would require more explicitness than is otherwise desirable. The proliferation of machines with lousy floating-point hardware – together with the misapprehension that the automatic computer is primarily the tool of the numerical analyst – has done much harm to the profession."
>
> Edsger W. Dijkstra [1]

> "The maxim 'Nothing avails but perfection' may be spelt shorter, 'Paralysis'."
>
> Winston S. Churchill [2]

After more than three years' deliberation, a subcommittee of the IEEE Computer Society has brought forth a proposal [3, 4, 5] to standardize binary floating-point arithmetic in new computer systems. The proposal is unconventional, controversial and a challenge to the implementor, not at all typical of current machines though designed to be "upward compatible" from almost all of them. Be that as it may, several microprocessor manufacturers have already adopted the proposal fully [6, 7, 8] or in part [9, 10] despite the controversy [5, 11] and without waiting for higher-level languages to catch up with certain innovations in the proposal. It has been welcomed by representatives of the two international groups of numerical analysts [12, 13] concerned about the portability of numerical software among computers. These developments could stimulate various imaginings: that computer arithmetic had been in a state of anarchy; that the production and distribution of portable numerical software had been paralyzed; that numerical analysts had been waiting for a light to guide them out of chaos. Not so!

Actually, an abundance of excellent and inexpensive numerical software is obtainable from several libraries [14-21] of programs designed to run correctly, albeit suboptimally, on almost all major mainframe computers and several minis. In these libraries many a program has been subjected to, and has survived, extensive tests and error-analyses that take into account the arithmetic idiosyncrasies of each computer to which the program has been calibrated, thereby attesting that no idiosyncrasy defies all understanding. But the cumulative effect of those idiosyncrasies and the programming contortions they induce imposes a numbing intellectual burden upon the software industry. To appraise how much that burden costs us we have to add it up, which is what this paper tries to do.