An Interview with the Old Man of Floating-Point

Reminiscences elicited from William Kahan by Charles Severance

20 Feb. 1998

This interview underlies an abbreviated version to appear in the March 1998 issue of *IEEE Computer*.

Introduction

If you were a programmer of floating-point computations on different computers in the 1960's and 1970's, you had to cope with a wide variety of floating-point hardware. Each line of computers supported its own range and precision for its floating point numbers, and rounded off arithmetic operations in its own peculiar way. While these differences posed annoying problems, a more challenging problem arose from perplexities that a particular arithmetic could throw up. Some of one fast computer's numbers behaved as non-zeros during comparison and addition but as zeros during multiplication and division; before a variable could be used safely as a divisor it had to be multiplied by 1.0 and then compared with zero. But another fast computer would trap on overflow if certain of its numbers were multiplied by 1.0 although they were not yet so big that they could not grow bigger by addition. (This computer also had nonzero numbers so tiny that dividing them by themselves would overflow.) On another computer, multiplying a number by 1.0 could lop off its last four bits. Most computers could get zero from X - Y although X and Y were different; a few computers could get zero even though X and Y were huge and different.

Arithmetic aberrations like these were not derided as bugs; they were "features" of computers too important commercially for programmers to ignore. Programmers coped by inventing bizarre tricks like inserting an assignment " X = (X + X) - X " into critical spots in a program that would otherwise have delivered grossly inaccurate results on a few aberrant computers. And then there were aberrant compilers ... .

"Reliable portable numerical software was becoming more expensive to develop than anyone but AT&T and the Pentagon could afford. As microprocessors began to proliferate, so did fears that some day soon nobody would be able to afford it."

Some History

In 1976 Intel began to design a floating-point co-processor for its i8086/8 and i432 microprocessors. Dr. John Palmer persuaded Intel that it needed an arithmetic standard to

prevent different boxes with "Intel" on the outside from computing disparate results inside. At Stanford ten years earlier, Palmer had heard a visiting professor, William Kahan, analyze commercially significant arithmetics and assess how much their anomalies inflated the costs of reliable and portable numerical software. Kahan had also enhanced the numerical prowess of a successful line of Hewlett-Packard calculators. Palmer, now the manager of Intel's floating-point effort, recruited Kahan as a consultant to help design the arithmetic for the i432 (which died later) and for the i8086/8's upcoming i8087 coprocessor.

"Intel had decided they wanted really good arithmetic. I suggested that DEC VAX's floating-point be copied because it was very good for its time. But Intel wanted the `best' arithmetic. Palmer told me they expected to sell vast numbers of these co-processors, so `best' meant `best for a market much broader than anyone else contemplated' at that time. He and I put together feasible specifications for that `best' arithmetic. Subsequently Silicon Valley heard rumors (not from me) about the i8087. They were aghast; how could Intel pack all that into a chip with only several thousand transistors?"

"I have said from time to time, perhaps too cynically, that other Silicon Valley companies got worried enough to join a committee that might slow Intel down. It was the committee assembled to produce a standard for floating-point arithmetic on microprocessors."

IEEE p754

Anarchy, among floating-point arithmetics implemented in software on microprocessors, impelled Dr. Robert Stewart to convene meetings in an attempt to reach a consensus under the aegis of the IEEE. Thus was IEEE p754 born. The second meeting was held one evening in November 1977 at a San Francisco peninsula hotel under the chairmanship of Richard Delp. It attracted over a dozen attendees and Prof. Kahan.

"These guys were serious. Some had been sent by microprocessor makers who had floating-point implementations in mind. National Semiconductor sent two. Zilog sent someone thinking about a Z8070 for its Z8000. Motorola was represented by Joel Boney who then led their project in Austin, Texas, to build what later (1984) became the MC68881/2 coprocessors. These were beautiful jobs. Of course, Motorola and Zilog had a lot more transistors to play with by then."

"Most mainframe makers, like CDC and Cray, sent nobody to these meetings, construing them to matter only to microprocessor makers. IBM was there only to observe; they knew their microprocessors were coming but couldn't say much."

After that 1977 meeting Kahan went back to Intel and requested permission to participate in the standard effort. With permission granted, Kahan and his student Jerome Coonen at U.C. Berkeley, and a visiting Prof. Harold Stone, prepared a draft specification in the

format of an IEEE standard and brought it back to an IEEE p754 meeting. This draft was called "K-C-S" until p754 adopted it.

"I got Palmer's verbal permission (which I think is very much to Intel's credit) to disclose specifications for the i8087's non- transcendental functions but not its architecture. I could describe the precisions, exponent ranges, special values (Not-a-Number and Infinities), and storage formats (which differed from the VAX's). I could also disclose some of the reasoning behind our decisions: WHY but not HOW. And I had to bite my tongue rather than say a word about the i8087's transcendental functions or its peculiar architecture."

(Most math libraries have exponential, logarithmic and trigonometric transcendental functions, and decimal <--> binary conversions.)

"We really didn't want to give away the whole ball of wax. Intel was about to spring a real surprise on the world:- one chip with MOST of the essentials of a math library. We wanted to put ALL of it on one chip but we had only 40,000 transistors."

Early in the Process

Prof. Kahan presented the K-C-S draft to the IEEE p754 working group. It had several proposals to consider. DEC was advocating that their formats be adopted. Other proposals came from microprocessor producers. The initial reaction to the K-C-S document was mixed.

"It looked pretty complicated. On the other hand, we had a rationale for everything. What distinguished our proposal from all the others was the reasoning behind every decision. I had worked out the details initially for Intel. My reasoning was based on the requirements of a mass market: A lot of code involving a little floating-point will be written by many people who have never attended my (nor anyone else's) numerical analysis classes. We had to enhance the likelihood that their programs would get correct results. At the same time we had to ensure that people who really are expert in floating-point could write portable software and prove that it worked, since so many of us would have to rely upon it. There were a lot of almost conflicting requirements on the way to a balanced design."

"Also the design had to be feasible, and not just in microcode; I had to be reasonably confident that, ultimately, when fast floating-point arithmetic was wired into hardware it would still run at a competitive speed. At the same time I had to be discreet; there were goings on at Intel that I couldn't disclose to the committee. These were certain implementation tricks and details that we had contemplated with a view to a future beyond the i8087. This applied particularly to Gradual Underflow,-- the subnormal numbers. Gradual Underflow became a real bone of contention. I had in mind a way to support Gradual Underflow even at the highest speeds, but I couldn't talk about that!"

"I was free, of course, to reveal implementation methods already known even if not widely known. An example was how to compute SQRT in software correctly rounded at a cost scarcely worse than people were obliged to pay for an incorrectly rounded SQRT; this information had to be supplied for National Semiconductor because they had no room on their chip for SQRT hardware. Without revelations of this sort, the IEEE p754 committee could not have endorsed arithmetic specifications so stringent as would previously have been deemed unfeasible. Still, reticence about truly arcane implementation details may have prolonged the committee's disputes over unfamiliar aspects of the K-C-S draft more than we appreciated at the time."

Quickly, the choice narrowed down to two proposals. The existing DEC VAX formats, inherited from the PDP-11, had the advantage of a huge installed base. But DEC's original double precision `D' format had the same eight exponent bits as had its single precision `F' format. This exponent range had turned out too narrow for some double precision computations. DEC reacted by introducing its `G' double precision format with an 11 bit exponent that had served well enough in CDC's 6600/7600 machines for over a decade; K-C-S had chosen that exponent range too for its double precision.

With its `G' format, DEC's VAX disagreed with the K-C-S draft primarily in their treatments of underflow, which the VAX flushed to zero instead of handling gradually. Consequently their exponent biases and over/underflow thresholds differed; K-C-S let numbers grow twice as big as VAX did without overflowing. Exponent biases differed by 2 . If only K-C-S exponents could have been reduced by this picayune difference, all arithmetics conforming to IEEE 754 would now use the VAX's formats, much to DEC's advantage. Why didn't IEEE 754 go that way?

Gradual Underflow becomes a Big Issue

Until the 1980s, almost all computers flushed underflows to zero and almost all programmers ignored them. In fact, Crays had no practical way to detect them and VAXs went likewise by default. Software users had no recourse but to choose different data if underflow caused their software to malfunction and they noticed it. Noticed malfunctions were uncommon, and most of these could be traced to a chasm poked into the number system by flushing numbers smaller than the underflow threshold to zero. This minuscule chasm between zero and the smallest positive normalized floating-point number yawned many orders of magnitude wider than the gaps between adjacent slightly larger floating-point numbers. An experienced numerical analyst could find ways around the chasm but naive programmers learned about it the hard way.

How many victims fell into the chasm? Nobody knew. Anecdotal evidence from university computing centers using DEC computers with 8-bit exponents pointed to at least a victim per month per machine during the 1970s. What would happen when

computers became over a thousand times more numerous and arithmetic over a thousand times faster?

"Gradual Underflow diminished the risk of falling into the chasm by filling it with `subnormal' numbers separated by the same gap as separated the smallest positive normalized number from its next larger neighbor. This implied that almost all (not all) underflows would get buried among subsequent rounding errors of which the programmer had to take account anyway. In short, programmers could keep on ignoring underflow, but now with less risk, so long as underflow was gradual. It also got rid of the anomalous possibility that predicates ` X==Y ' and ` X-Y == 0.0 ' might disagree for finite (tiny) operands X and Y . But Gradual Underflow was no panacea and it wasn't cost free."

The primary objection to Gradual Underflow arose from a fear that it would degrade the performance of the fastest arithmetics. Microcoded implementations like the i8087 and MC68881/2 had nothing to lose, but hardwired floating-point, especially if pipelined, seemed slowed by the extra steps Gradual Underflow required even if no underflows occurred. Two implementation strategies were discussed during meetings of the p754 committee. One used traps, one trap to catch results that underflowed and denormalize them to produce subnormal numbers, and another trap to catch subnormal operands and prenormalize them. The second strategy inserted two extra stages into the pipeline, one stage to prenormalize subnormal operands and another to denormalize underflowed results. Both strategies seemed to inject delays into the critical path for all floating-point operations, slowing all of them.

The dispute over Gradual Underflow swirled around its benefits versus its costs, as if its delays could not all be eliminated. Perhaps some cards were being held too close to chests.

"Now that my confidentiality obligations to Intel have expired, I can discuss ways to implement Gradual Underflow in hardware without delaying all floating-point operations. One approach exploits the mechanisms processors use to cope with cache management. Whenever an event like a cache-miss, underflow or subnormal operand is detected, what I shall call a " cliche' " can be inserted into the schedule of partially decoded instructions awaiting execution. A cliche' changes no dependencies among instructions in the schedule, requires at most a register or two to be set aside for it in the architecture, and may need some special operations added to the instruction set. Underflowed results must be denormalized; subnormal operands must be prenormalized before multiplication or division; both these operations are already stages in floating-point addition's pipeline, and can be simplified if floating-point registers carry a few extra bits. Generally, a cliche' serves a function similar to what DEC's "PAL-code" does now for its Alpha, a chip with very speedy IEEE 754 arithmetic in it."

Another approach, pioneered in an early MIPS chip, lightens the burden of trap handling software by ensuring that the floating-point pipeline will be empty when a trap occurs.

This is done by inhibiting the issue of subsequent floating-point operations until after operands with exponents too extreme to be guaranteed free from all risks of over/underflow have emerged from the pipeline. The detection of those extreme exponents is synchronized with the detection of cache misses in order not to further delay the processing of unexceptional instructions which constitute the majority by far.

The Battle over Gradual Underflow

Like a religious war it went on for years, amidst which Delp passed the burden of chairmanship to David Stevenson, then at Zilog. DEC's numerics expert Dr. Mary Payne led the opposition to K-C-S. To win the battle one side had to win near unanimity among the members of p754. Packing the committee was a temptation to be resisted.

"At an early meeting of p754 in the late 1970s a hardware engineer from DEC had stated flatly that K-C-S could not be built to run as fast as VAX arithmetic hardware. Meanwhile George Taylor, then a graduate student supervised by Computer Science Prof. Dave Patterson at U.C. Berkeley, had been building K-C-S floating-point onto two accelerator boards for a VAX there. The plan was to substitute these for DEC's without changing the VAX instruction set, and then see how well software ran on each arithmetic. After Taylor showed his design to a p754 meeting nobody could dispute the feasibility of fast K-C-S floating-point. Taylor subsequently built K-C-S floating- point into the ELXSI 6400, the first super-minicomputer, which out- performed the most expensive DEC VAX systems for a while."

"(Dr. Taylor is now a principal at Exponential. His boards for the VAX stayed uninstalled because David Cary, the student in charge of final assembly and test, was killed by a tragic swimming mishap that disheartened his collaborators, not to mention his family.)"

"The claim that K-C-S was too hard to implement in fast hardware had been undermined, and so had some of DEC's credibility. They turned next to an attempt to prove from a theoretical standpoint that Gradual Underflow was bad for numerical software. If this could be proved, a chain of logic would lead K-C-S inexorably to the same exponent bias as in DEC's VAX, which could than accommodate the other details of K-C-S rounding and exception handling by small architectural tweaks."

"This attempt had some merit because the troubles caused by underflow cannot all be alleviated by Gradual Underflow; it dispatches only those underflows that it can render practically indistinguishable from roundoff. The same effect can be accomplished by a knowledgeable and diligent programmer at the cost of a few extra tests against artfully chosen thresholds. Some such tests may be needed even with Gradual Underflow to render a program invulnerable to over/underflow. How big a burden should hardware designers bear to save adept programmers from a few tests, and to protect some unknowable number of users of naively written programs from falling unwittingly into a tiny chasm?"

The Showdown

"Support was accreting to the K-C-S draft not just from my students Jerome Coonen, Jim Demmel, Peter P-T. Tang and George Taylor, but also from ex-students like Dr. David Hough then at Apple, from my friends in industry like Dr. Fred Ris at IBM and Dr. W. Jim Cody at Argonne National Labs, and many more. The cogency of their prompt responses to technical questions convinced doubters that the K-C-S draft's details really were necessary and coherent. Their support was crucial also because they, especially Coonen, were far more astute politically than I, who still cannot understand how this country ever elected a few of its recent Presidents."

Despite endorsements of the K-C-S draft by luminaries like Stanford Prof. Donald Knuth and the English doyen of error-analysis Dr. J.H. Wilkinson, arguments over Gradual Underflow and occasionally other details continued to prolong meetings of p754 into the wee hours of many a morning.

"DEC tried to break the impasse by commissioning Univ. of Maryland Prof. G.W. (Pete) Stewart III, a highly respected error-analyst, to assess the value of Gradual Underflow. They must have expected him to corroborate their claim that it was a bad idea. At a p754 meeting in 1981 in Boston Pete delivered his report verbally: on balance, he thought Gradual Underflow was the right thing to do. (DEC has not yet released his written report so far as I know.) This substantial setback on their home turf discouraged DEC from continuing to fight K-C-S in meetings of p754."

From that time on the standard ground slowly towards completion after several changes in wording and a few compromises. One of these brought Signaling NaNs into existence to permit dissenters to augment what they considered too parsimonious a set of special operands consisting of signed zeros and infinities and NaN (Not-a-Number) in the K-C-S draft. Finally p754 sent the draft up to the IEEE Microprocessor Standards Committee.

"And there it sat for over a year as if in Limbo. Some kind of back- room politicking must have gone on, I guess. It is easy to imagine someone arguing that IEEE p754 was not a standard but a design, and an unproven one. Events soon overtook that argument. By 1984, p754 had been implemented by Intel, AMD, Apple, ELXSI, IBM, Motorola, National Semiconductor, Weitek, Zilog, AT&T, ... I lost count."

"Those pioneers deserve our admiration because IEEE 754 is difficult to build. Its interlocking design requires every micro-operation to be performed in the right order or else it may have to be repeated. Maybe some implementations conformed only because their project managers had ordered their engineers to conform as a matter of policy and then found out too late that this standard was unlike those many standards whose minimal requirements are easy to meet. Anyway, a year before it was canonized, IEEE 754-1985 had become a *de facto* standard, the best kind."

Afterthoughts

"I did not bill Intel for consulting hours spent on those aspects of the i8087 design that were transferred to IEEE p754. I had to be sure, not only in appearance and actuality but above all in my own mind, that I was not acting to further the commercial interest of one company over any other. Rather I was acting to benefit a larger community. I must tell you that members of the committee, for the most part, were about equally altruistic. IBM's Dr. Fred Ris was extremely supportive from the outset even though he knew that no IBM equipment in existence at the time had the slightest hope of conforming to the standard we were advocating. It was remarkable that so many hardware people there, knowing how difficult p754 would be, agreed that it should benefit the community at large. If it encouraged the production of floating-point software and eased the development of reliable software, it would help create a larger market for everyone's hardware. This degree of altruism was so astonishing that MATLAB's creator Dr. Cleve Moler used to advise foreign visitors not to miss the country's two most awesome spectacles: the Grand Canyon, and meetings of IEEE p754."

"Programming languages new (Java) and old (Fortran), and their compilers, still lack competent support for features of IEEE 754 so painstakingly provided by practically all hardware nowadays. S.A.N.E., the Standard Apple Numerical Environment on old MC680x0-based Macs is the main exception. Programmers seem unaware that IEEE 754 is a standard for their programming environment, not just for hardware."

"If better precision, directed roundings, and exception handling capabilities latent in hardware but linguistically inaccessible remain unused, their mathematically provable necessity will not save them from atrophy. The new C9X proposal before ANSI X3J11 is a fragile attempt to insinuate their support into the C and C++ language standards. It deserves the informed consideration of the programmers it tries to serve, not indifference."

"In the usual standards meetings everybody wants to grandfather in his own product. I think it is nice to have at least one example -- IEEE 754 is one -- where sleaze did not triumph. CDC, Cray and IBM could have weighed in and destroyed the whole thing had they so wished. Perhaps CDC and Cray thought `Microprocessors? Why should we worry?' In the end, all computer systems designers must try to make our things work well for the innumerable (innumerate ?) programmers upon whom we all depend for the existence of a burgeoning market."

Epilog

The ACM's Turing award went to Kahan in 1989. IEEE Standard 754- 1985 for Binary Floating-Point Arithmetic is up for reconfirmation this month.