Reflections on the Intel 8087 Math Coprocessor

William Kahan, Emeritus Prof. UC Berkeley
Recorded 10 Sept 2025 in Berkeley, CA

So, first of all, I'm Prof. William Kahan and, of course, I'm retired, And the only reason that you might listen to what I have to say is that I started with computers in 1953. That's when I learned that they existed at the University of Toronto, and that's when I read things like a computer design of arithmetic That experience turned out to be valuable in 1964 when IBM had come out with its family of computers that used hexadecimal arithmetic instead of the binary arithmetic that had been their forte earlier.

The hexadecimal arithmetic that was designed by somebody who there felt that he was entitled to cut corners, for example. to just disregard what we call a guard digit. And that caused various anomalies in the arithmetic of the computer that IBM wanted to sell us. We discovered that that computer wouldn't run the software, even the software that had run on the 7090 and the 7094, because the arithmetic was bizarre. And so I had joined a committee of SHARE, the user group. There were about three or four of us on that committee, and our job was to recommend what changes we wanted IBM to adopt. And, in fact, they adopted most of those changes.One example was producing a guard digit in add/subtract, and doing a number of other things. That's why I earned my right to have people listen to me about the issues of computer arithmetic, which had just been a hobby before.

There were many anomalies in computer arithmetic because everybody in any different company would implement floating point differently. And, of course, the floating point would be used by the scientists and engineers and statisticians, regardless of its arithmetic anomalies, because they didn't know about them.So then I accumulated experience in computing arithmetic’s diversity. For example, the Burroughs B5000. That Burroughs had octal arithmetic, and it had its own peculiarities, and all sorts of other computers had binary arithmetic, but the binary arithmetic in one computer was incompatible with the binary arithmetic in another, either because the word sizes were different or because the different computers did multiplications differently, or did the additions differently, actually subtractions.

So that was why I had reason to join the committee of the IEEE that was trying to standardize arithmetic. And I think I joined that committee in 1977. Well, whatever it was, previously I had done some consulting for Hewlett Packard on their calculators. And that's only relevant to this discussion because John Palmer learned that I was doing consulting, and so he thought that I could do consulting for his project at Intel. His idea, John Palmer's idea, was to produce a coprocessor, which would have not only floating point arithmetic of one or two precisions, well it actually turned out three. But to have also the entire math library, which would normally have been sold with a compiler, different compilers with different libraries. John wanted to have not only a good arithmetic, but also a good library.

Because he had listened to one of my lectures at Stanford, when he was a graduate student, when he learned that I was consulting for Hewitt Packard, he invited me to consult with Intel on the floating point arithmetic in a coprocessor chip. I said, well, you know, IBM's arithmetic could be mimicked, it's tolerable now. because of the SHARE committee. That was the IBM mainframe user group. The SHARE committee had tidied up the arithmetic. It was tolerable, and he'd have a large market. And John said, "No, we want a good arithmetic." And I said, "W, the DEC Vax has a very good binary arithmetic. You could mimic that, and now he said, "We want the best." And so I helped to design what would be the best arithmetic. And we got almost all of that on the chip.

The math library was another matter. Binary-decimal conversion, of course, had to be assisted. And so that was built into the chip. But the library for the elementary transcendental functions, ah, that was a different problem. We couldn't get all of sine, cos, tan, and arcsine, arccos, arctan, and the hyperbolic functions. We couldn't get all of that into the chip. There just wasn't going to be space. So what we had adopted instead was to have a small sample of functions from which the elementary transcendental functions could be computed very quickly in software. That was what it amounted to: the tan half-angles and arctan half-angles, and similar functions, hyperbolic functions, for the computations then of logarithms, and exponentials. And that, we figured, would fit in the chip.

Well, we were very optimistic. It turned out that there wasn't enough room on the chip, unless something very elaborate would happen to enhance the space for the microcode.

We would have to have more space for the microcode. In consequence, the people in Santa Clara, they thought that it couldn't be done. There would not be enough space on the chip. But the guys from Intel at Haifa, they thought maybe they could do it. And in due course I met Rafi Nave from Intel Israel. I think I had two discussions with him at home here. He got the idea of why we wanted these things and decided that he'd find a way to do it.

Well, what they did was figure out how to get two bits per transistor instead of one in the read-only memory. It was a question of having a somewhat elaborate driver circuit for driving the memory. But once they figured out how to do that, and it was somewhat miraculous, then they were able to get everything that we wanted on the chip, even if there would be a certain amount of software assistance to get the actual sine, cos, tan, etc. from what I described as kernel functions like the tan half-angle, arctan half-angle, and so on, that could go on the chip. But then, it was just a very small amount of software that would elaborate the other functions. So that was something.

Let me put it this way, I'm convinced that somebody in Santa Clara leaked the fact that we were going to put all that stuff on the chip. I don't know who leaked it. but that is what attracted various other companies' engineers to stay late, after we had discussed issues about the IEEE standard. They stayed late to find out how to do it.

I had persuaded Intel that Intel should release the arithmetic, just the arithmetic: add, subtract, multiply, divide, and square root. Just release what we had done there, and we'd say nothing in that committee about all the other functions that we were going to go on the ship. And that attracted engineers who had not learned this in their studies. They had not learned how to do floating point arithmetic. Integer arithmetic? Yes, they could do that. But floating point arithmetic had little subtleties.

And what we got into the standard was not only a very clean version of arithmetic, but directed roundings, so that we could support what was called interval arithmetic if somebody wanted it. Although hardly anybody wants it. And we had something called *tempreal* in the Intel version.

Now it's time to get into itty bitty details. Single precision, a 32 bit word with 24 significant bits. Double precision. This is a 64 bit word, and it had 53 significant bits. But

tempreal was something that could be expanded. And the reason that we had to have tempreal was that I had asked, what is the widest word which will accommodate a carry propagation for add? And I was told it was 67 bits. Well, 67 bits would be enough to compute double precision transcendental functions fairly accurately within a unit in the last place. But to support 67. Yes. For tempreal, it was 64 significant bits, but the programmer was advised that it could grow wider. The intention, and the term tempreal reflects that, the intention was that that word would ultimately fit into a 128 bit wide word. Well that turned out to be quadruple precision, which was implemented later. That had the same exponent as tempreal, 15 bits. Plus the sign. Right. So we had 16 bits for the epopnent and sign. And the rest would grow to fill up a word ultimately 128 bits wide. And you could use that at your peril, because if you don't program it in a way that it allows for subsequent growth, then your program may be rendered obsolete.

But it was not necessary to use the tempreal format. We used it internally in the chip. And it was available for somebody to use if they wanted it, but it was really intended to support the single precision and double precision formats. And those were supported in various languages. Some would support both of them. Some would support only the double precision format. For example, Java supported only double precision initially. So you could support single or single and double, or just double, but if you wanted to use tempreal, then you should not depend upon how wide that word would be, because it could change. That was the intention. And that intention was justified, at least in my mind, because I expected that ultimately, the carry propagation would grow from 67 bits to perhaps something on the order of 120. Ultimately, well, that actually never materialized. A chip never was produced to do that. And that's okay. At least we had the standard of single precision, double precision, and whatever was wider. That all was standardized. Also standardized were directed roundings, including an ordinary rounding. It had a guard digit and a sticky bit to ensure that the result of subtract, multiply, divide, and square root would be a correctly rounded value. Now, we didn't produce something that was always correctly rounded for sine, cos, log, etc. That would be accurate to less than one unit in the last place, but not just half a unit of the last place.

That's something to be discussed elsewhere, because Muller, in Lyon, and his students found that they could get correctly rounded elementary transcendental functions, but it would cost triple precision to do that occasionally. And that's something that you could

build into your library if you want to support it. But supporting it on a vectorized machine would be a bit of a stretch. Let's say, I want a double precision cosine of every single element in a vector in a vectorized machine. Well, in order to get correctly rounded, you would perhaps have to compute it to triple precision occasionally, just occasionally. And to do that, you would have to interrupt the vector elements in order to get them correctly rounded in every instance. And so I thought that that was too much to build into one chip. But some people would disagree with me. That's okay. But there we were.

That was the 8087 chip. For the 80387 coprocessor for the 80386 CPU, the demand for rapid Fourier transforms justified building sine and cosine into the chip, the 387. So what would have been done in software on the 8087 was instead done on the 387 chip.

Actually, we really have to acknowledge that, for the 8087, it was not only the development of two bits per transistor in the microcode. Also, the design of a barrel shifter was a significant advance over the ordinary shift structure. The ordinary shift structure was somewhat slow, because you would be able to just count the shifts. as one bit at a time, as you run your loop, but the barrel shifter said in effect that you could shift in jumps. The jumps provided by the barrel shifter's logic, where you would be able to shift one bit, two, four, eight, bits at a time. That was a significant contribution to the performance of the 8087 and it also propagated to the 387 and others.

The 487, that was different. It was now something that you would buy already on the CPU chip. And for what would have been the 587, but actually Pantium, that arithmetic also would be different and built into the chip. But the Pentium had an array multiplier. Instead of doing multiplication essentially one bit of a time, it could do multiplication in an array that would not have been able to fit into the previous chips. But that really gets far away from what we're talking about.

So the architecture of the 8087 persisted to the 387 and what amounted to a 487 that had a brief lifetime. It was now something that would be standardized until the advent of artificial intelligence, where people have chosen half precisions. Well, a half precision word is somewhat of a conundrum now, because how do you choose the exponent width? Either you have a narrow range, or if you have a wider range, you have less precision. And that's something for somebody else to worry about.