A.M. Turing Award
Interview with William Morton Kahan
1989 ACM Turing Award Recipient
Thomas Haigh, Interviewer

TH = Thomas Haigh, Interviewer
WK = William Morton Kahan, 1989 ACM Turing Award Recipient

TH: I'm Thomas Haigh and this is an oral history interview conducted with Professor William Kahan. It's the 12th of March, 2016, and we're in Professor Kahan's house in Berkeley, California.

Professor Kahan, thank you very much for taking part in this interview, which is part of the ACM's series of video oral history interviews with Turing Award winners.

I should also mention, as you know, we did an interview back in 2005. That ran to almost 24 hours of tape and was very detailed. So what we'll be doing here is a shorter, more accessible interview format where we'll be recapping for a broad audience some of the key points of your career.

To begin with, I wonder if you can tell us something about your early life, education, and family background.

WK: I was lucky to have parents who valued education very highly. It had been denied them in their European homelands – Poland and what is now Moldova. They sacrificed a good deal of their comfort and pleasures to put us into Forest Hill Village, which was a sub-municipality in Toronto that had reputedly the best public school system. Our house was three doors from the boundary of this Forest Hill Village. Our backyard was separated from the railway tracks by a creek, so we were sort of out in the boonies of Forest Hill Village. But that really made us, my brother and me, made us what we are.

It was a terrific education and, I thought to give you some idea, I got skipped two grades. When I entered Grade 9, I had skipped Grade 8 and I forget which grade before it. So they thought I might not be able to handle the rigors of an academic program and put me into the tradesman's program. Which meant that while some of my contemporaries were taking classes in Latin, I was taking classes in shop. I was learning how to use saws and drill presses and lathes and a little bit of brazing, a tiny bit of welding. And that suited me perfectly. I loved it.

And in fact I've got something here that I made when I was 12 years old. I've taken off a little cap that prevents it from scratching my shirt. Now you see, this looks like a screwdriver, doesn't it?

TH: It does.

WK: But it's not. It's three screwdrivers. If I unscrew it here, I then take out… there's another screwdriver. And if I unscrew that, why there's another screwdriver. Really handy for tightening the screws on my glasses, which I'm not wearing today.

I've shown you this because it will give you some idea that although I'm a mathematician, I was pretty handy with my hands and that is the way I've been all my life. Repair my own household appliances, usually can repair the plumbing, the cars, and so on. I don't think of life as something that should be lived just as a mathematician or just as an engineer. And I learned that at school. Well, after Grade 9, they realized I could handle the academic program, so they stuck me into Grade 10 with the academic class, and I had to catch up on Latin. I loved Latin. It's a very epigrammatic language, very different from my normal style. And this interest continued when I entered university. Now I won a scholarship…

TH: Let's backtrack a little bit with the family. So your parents moved to, emigrated to Canada?

WK: Oh, they emigrated to Canada in the '20s in order to escape pogroms and really vicious anti-Semitism in Poland, which is a foretaste really of what was to come when Germany invaded. They were in many ways ahead of their time. My mother designed dresses for girls to work as receptionists or even female doctors, which were becoming more and more numerous. And my father ran the factory that made the dresses, and he bought the cloth and went out to sell them. They were ahead of their time because, for instance, they refused to use the epithets that are sometimes used now. We were not to use words like "nigger" or "shiksa" or disparaging terms for Italians or whatever.

[phone rings] Oh, excuse me. I think Sheila will get that. Yes.

The idea that women were disadvantaged in the workplace was planted firmly because that happened to my mother. That's why she ended up working with my father rather than for other, more established outfits.

Because they valued education, as I said, we went to this very good school system. There was a particular friend, David Gauthier, who was the son of one of the teachers there. That teacher is the one who, among other things, taught shop. That's where I made this[1]. His son, who was in the same year as me there, he was outstanding, just a marvel to behold. And after Grade 13, which meant 13 in Canada at the time instead of 12 that you have here, he won the province's most prestigious scholarship, and I think I got the second or third. But it wasn't tenable in a professional school, so that meant that I couldn't use it to go to engineering school and instead decided, "If the government's going to pay my way, why don't I learn something hard like mathematics?"

TH: So you think otherwise you would have gone to engineering school?

WK: Oh yes. Well, that was my mindset at the time. I expected to be an electronic engineer because that was my hobby.

[1] Showing his set of small nesting screwdrivers.

TH: Yes. So you've already spoken about enjoying crafting mechanical things. What was your first exposure to electronics?

WK: Oh. Well, when I was 11, I wound wire around a cereal box and attached it with a tuning capacitor and a little crystal – the crystals in those days had these little feelers so you can find a sensitive spot – and had earphones and a long wire antenna. That was the first radio that I built. But I'd already understood about electricity and I read a couple of books so that I became well-acquainted with things like the vacuum tubes of that era and lots of other electronic and electrical things. But that was just a hobby. That was just something I did for fun. I used to repair things around the house, repaired my bicycle quite often. So when I entered college…

TH: Now we'll just give people a sense of the chronology.

WK: Say again?

TH: Just to give people a sense of the chronology, I believe that was in 1950.

WK: That's correct. The autumn of 1950. That was also the year when I suggested to Sheila that she might keep open the possibility that we might get together more firmly, but I didn't want her to feel trapped. I didn't say, "We'll go steady" or anything, just let her know I'm interested.

TH: So had you been in the same high school?

WK: We'd been in the same grade school briefly, but she went to a different high school than I did. We did go to the same Jewish synagogue youth group. And she was gorgeous. She just didn't know it. And she was genuine and didn't try to pretend that she was silly or helpless, as so many girls did. Their mothers told them, "Don't let the boys know how clever you are or intelligent, because …." Sheila wasn't like that.

And when I started in college, I started in math, physics, and chemistry, as did David Gauthier. But he decided that wasn't for him and he switched to philosophy. About 175 people started out in the first class. It was a calculus class. Of those, 27 graduated in math, physics, or chemistry. The others had to go somewhere else and take other classes. That's an attrition ratio that nowadays would be considered scandalous, but then it was considered quite normal.

Well, it was shortly after that that I got a summer job. I think that was the summer of '51. I'm not quite sure. It may have been the summer of '52. I got a job through what amounted to a classroom acquaintance whose father owned this electronics company that made electrical instruments, and they were repairing mainly high-frequency direction-finding sets that were in use by the then Royal Canadian Navy. They were also repairing various instruments for the Air Force. I was working next to a bunch of immigrants of whom only one knew Ohm's law. The others knew nothing about electricity. They were just told, "Here, replace that part," and they had a soldering gun and soldering iron and so on. The foreman didn't know anything about electricity. He just took his orders from who knows who.

But then they found out that I understood this subject and I was able to produce what are now called "fault trees," so that when one of the instruments came in from the Air Force and had been damaged somehow, it was just a sequence of tests or questions that you'd ask, and on the basis of that, you could pretty well identify the part that had to be replaced or recalibrated. And that saved an enormous amount of time because what they had done previously was replace everything that looked a little bit darker than it should be. I was even able to repair the moving coil meters. You know, these things that have a hand that swings back and forth in response to the voltage. My eyesight was good enough and hands were steady enough that I could repair those, and that came in handy later.

In my third year in college, I got preoccupied with Sheila. She had very reasonable doubts, and I had to overcome those doubts to persuade her that we should try to grow old together. And I couldn't think of anything else that year, so I really made a mess of it academically.

TH: Now did Canada at that point have three-year degrees under British system, or was it four-year?

WK: No, it was four-year. It was a four-year degree. This was my third year and I really did louse it up. I think I've mentioned that I may have spelled my name correctly on the thermodynamics exam, but if so, that was the only thing I got right. But I also took the Putnam exam in I guess it was December of '52, and I scored fairly high on that. I even won a little pin. That's the way my mother found out that I had other interests, when I gave the pin to Sheila instead of to my mother.

Well, so I figured out that it's not enough to love someone. If you would like to be loved, you have to deserve it. So from becoming the laziest man on campus, I turned into someone industrious for my fourth year. Part of what I did was to fix the electrical and electronic equipment for the physics lab. I won't say that my contemporaries broke equipment so that I could fix it, but a lot of stuff did get broken and I fixed it. I also fixed electronic equipment for graduate students, mainly high-voltage pulse generators. I could fix the oscilloscopes and so on.

So that got me a little bit of money and I was I thought pretty happy. I'd made a choice to go into mathematics for the final year. Now that was done partly for a completely unreasonable reason. The professor who ran the optics lab that I took in my third year had noticed that I and my lab partner were getting superb results, much better than everyone else. My lab partner was Joe Vise[2]. He was a friend from high school. A guy with very similar inclinations – he also was good with his hands. So the first thing we did in the lab was to repair the equipment, and then of course we'd recalibrate it, and then we'd run the experiment. Of course our results in consequence came out rather better than those of people who didn't pay attention to these little details.

So this professor cornered me in the hall one Friday afternoon and said, "Kahan, you really ought to choose to go into physics next year. You know, all the world's greatest scientists have been physicists." As he said that, the name Pasteur went through my mind. I don't think he was a physicist. Neither was Darwin. And I got a whiff of his breath. Then I concluded that if being

[2] Joseph B. Vise, PhD in Physics earned in about 1960 in Brookhaven, if I remember correctly (added after the interview)

drunk at 3:30 on a Friday afternoon was what it meant to be a physicist, I wanted none of it, and chose mathematics. This is the way these very important decisions get made.

TH: Now am I correct that at this point in Canada, you wouldn't take a whole set of different courses at the same time, that you really would choose one subject for essentially the whole year?

WK: It wasn't a smorgasbord system, that's correct. You committed yourself to a certain program, there were certain courses you had to take, there was a small number of options. In my final year, one of my options I chose was a course in differential geometry and non-Euclidian geometry offered by Professor Coxeter[3]. Now Coxeter was the 20th century's greatest 19th century geometer I think, and also experienced in group theory. And I took the course because everything he did was so beautifully done. It was so elegant and crystalline. It wasn't that I wanted to be a geometer. I just wanted some of him to rub off on me.

TH: Are there any of the other professors or teachers that you would mention as particular influences?

WK: Well, there were teachers who influenced me, some in perverse ways. Ralph Stanton was an algebra professor. Very clever and very nasty. So I learned not to treat students the way he did. I mean I remember once having a conversation with him in which I expressed some sympathy for the downtrodden, and he said, "How can you care about people you've never met and don't know?" Well, that told me what sort of guy he was. But he went on. He moved from Toronto to Waterloo. He was largely instrumental in founding their college of mathematics and computer science, statistics, business mathematics, combinatorics, a flourishing enterprise. But he was a mean-spirited person and finally they got rid of him. They banished him to Winnipeg I think.

And there was Tutte, William Tutte[4]. He was very quiet and his sense of humour, these were always in-jokes, the jokes that only mathematicians or young mathematicians would appreciate. They were so sly. He was a delightful character and we did not know at that time what he had done during the Second World War.

TH: Yes. With the Fish code.

WK: This was, yes, the machine that the British called "Fish," the Germans called it the "Lorenz" machine. It was essentially a teletype machine on which you would type out your message but what got sent was encrypted, and at the other end when they received the encrypted message and

[3] Harold Scott MacDonald "Donald" Coxeter, FRS, FRSC (1907 – 2003) was a British-born Canadian geometer. Coxeter is regarded as one of the greatest geometers of the 20th century. He was born in London but spent most of his adult life in Canada. (from Wikipedia)

[4] William "Bill" Thomas Tutte, (1917 – 2002), was a British codebreaker and mathematician who was instrumental in breaking the German Lorenz code at Bletchley Park. After the War he completed a PhD at Cambridge and immediately took up an academic position at the University of Toronto. In 1962 he moved to the University of Waterloo where he stayed for the rest of his academic career.

punched up a tape and ran it through the tape reader, on a similar machine it would print out the plaintext. And apparently a message from… I don't know if it was France or Vienna was sent to someone in Holland, and the person in Holland complained that it had become corrupted and asked to have it sent again. And the British were eavesdropping and they picked up both messages. They could see there were similarities and they had some ideas about what the message might be about, so they had some ideas of what phrases or words might appear.

Bill Tutte was asked to look through this and see if he could figure it out, and he did. He figured out how the Lorenz machine worked without ever seeing one and they built a replica without ever seeing one. Now all they needed was the key. And to get the key, they built the Colossus. That was the motivation for this huge electronic computer, I think probably the first one built even if it didn't have a stored program.

Well, I had a number of other professors, but the one who influenced me most was unquestionably Kelly Gotlieb, Calvin C. Gotlieb[5]. That happened because in March… It was March of 1953 and I was told that there was a thing called a "computer." I asked, "What does this 'computer' do?" and somebody told me in brief outline, and it set me back. I thought, "I didn't think that was possible. How would they do a thing like that?" So I designed my own electrical computer using telephone relays. Of course I didn't build it. I used I don't know how many hundreds of telephone relays, and I know that at one point when I tried to compute how much power this thing would use, I came up with a number that was a large fraction of the electrical power generated by Niagara Falls. So it was a good thing that it was never built.

But I had it all drawn out on diagrams and I went in one day in I guess it was April to Kelly Gotlieb's office. He was the guy who was the de facto manager of the computer. The person officially in charge, Watson, was the head of the physics department, but Kelly Gotlieb was the one who actually ran things. I went into his office and I said, "Look, I'll show you mine if you show me yours." So he did. And he became first one of my professors and then a friend and a guru, and I've admired him greatly. When you ask who influenced me a lot, well, let's put it this way. There are many times when I would have been better off if I had taken his advice. And I think he's still alive in Toronto.

TH: Yes. In fact, there's a relatively recent oral history interview with him in the ACM Digital Library[6]. So I'm glad that they were able to get his experiences and memories documented.

WK: Oh good. Because he's played a vital role in initiating all sorts of activities using computers. I mean he was early on the scene for big data library stuff and so on on computers that couldn't hold much data. But he saw ahead.

[5] Calvin C. (Kelly) Gotlieb (1921 – 2016) has been called the "Father of Computing" in Canada. He received his MA in 1944 and his PhD in 1947 from the University of Toronto. In 1948, he was part of the first team in Canada assembled to design and construct digital computers and to provide computing services. In that year, he co-founded the original Computation Centre at the University of Toronto. He established the first university credit course on computing in Canada in 1950, and offered the first Canadian graduate courses in computing in 1951. In 1964, he founded the first graduate department of Computer Science in Canada, at the University of Toronto.

[6] http://dl.acm.org/citation.cfm?id=1370098

TH: Yes. So I believe the computer you're talking about here would be the FERUT?

WK: This was… Yes. The first computer in Toronto was actually built by Joe Kates[7]. He was an Austrian[8] refugee who came to Canada and was getting an advanced degree in electrical engineering. He and a sidekick, Len Casciato[9], they built a small computer to demonstrate that Joe had successfully figured out how to improve the read-around ratio. That's the analogue for cathode-ray storage of what we do now for DRAM, dynamic-read random-access memory. DRAM will lose its memory after a while if you don't refresh it, and in those days that's what you had to do with the cathode-ray memory. But also there was a problem. That was if you read a number from the cathode-ray memory, it would be done by really looking at secondary emission of electrons. The electrons emitted would splash into neighbouring numbers, and there was some very serious questions about what could you do to reduce the rate at which adjacent numbers would fill up with electrons that didn't really belong to them? Joe's thesis was how to improve this, how to improve this read-around ratio.

Although obviously in Toronto the people who knew about such things wanted a computer, but on the other hand, building a computer according to the design of this Austrian immigrant didn't seem like such a great idea to the powers that be, so they decided to go with something designed in the old country, in Britain, and that was the Ferranti Manchester Mark 1, the first machine that I think was intended to be sold commercially. That was the machine I cut my teeth on. I see that you've reproduced here parts of the Creed teleprinter code. Oh, it's all very familiar. These were the Creed teleprinter characters, and when you programmed this computer, you did it with these characters, because there was no assembly language.

TH: Yes. And there were five data channels on the tape…

WK: That's right.

TH: …so there were not an enormous number of characters available.

WK: Well, there were 32 characters available. The tape had five holes plus a sixth small hole for the sprocket. The tape reader was a marvel. It was an electromechanical marvel. It had a magnetic clutch. The clutch has two faces, one driven by a motor all the time, and the other one would drive the spindle for the tape reader. They were both immersed in a cylinder filled with oil and ground-up iron, iron powder. If there was no magnetic field, the driven disk would spin and it wouldn't have much effect on the other, which at its other end had a magnetic clutch to hold it still, you see. But when they wanted to move the spindle so that they could move the tape forward on its sprocket, they would un-magnetize the brake, magnetize the driven part, and then it would become solid. The magnetic particles would all clump together as if it was a solid connection.

[7] Here Kahan is not talking about FERUT but an earlier machine known as UTEC- the FERUT was a commercial machine made by Ferranti in England and purchased by the University of Toronto later. He does talk about FERUT in the following paragraph.

[8] Kahan originally said "Czech" but that was incorrect.

[9] For more information on the people and machines mentioned here consult *IEEE Annals of the History of Computing*, Vol. 16, Number 2, 1994 – a special issue on the history of computing in Canada.

This gadget would read 200 characters per second going all out. But the computer had a millisecond clock rate. It took a millisecond to do anything, three milliseconds to do a multiply. I think I'm the only person who ever got the tape reader to read integers, six-digit integers without stopping the tape. The tape flew through the reader and into a big bin, and if you got in the way, you could get hurt. And if there was any flaw in the tape, it would break. But alright, so there I am boasting about trivia, but it was an interesting machine.

TH: So if we step back a bit, can you give people a sense of what the machine was like physically? You described the tape reader. What else would be in the room?

WK: Well, there was a console. The console had a number of cathode-ray screens. Two of them were in effect… Oh, what is the word? Something that repeats what's on some other screen.

TH: "Slave"?

WK: Slaves. Yes, that'll do. They were just slaves that would tell you what was in the memory on two of the four cathode-ray screens that we had, and then there were two other screens that showed us the accumulator and the B registers. The B registers were Turing's idea[10]. They were ways of accessing indexed arrays by putting the index in a register, and then you put the address in your instruction – that's the address at the beginning of the array – and you add the contents to the register, except for one peculiarity. Although you added the contents of that register to the whole instruction, if you wanted to change the contents of the register, the only instruction available was the subtract instruction, because it turns out that two subtracts are like an add. Talk about RISC – you know, reduced instruction set computers.

TH: And that would have the advantage of eliminating the need to modify code in memory, because you can just change the …..

WK: Well, on other machines if you wanted to compute the address of an element in the array, you actually had to do the computation in the accumulator and then plant what it had computed into an instruction, and then you execute the instruction, that loads the accumulator with the thing you wanted to fetch. But thanks to Turing's idea, what you did was you loaded the instruction[11] with an address at the beginning of the array knowing that three bits of that instruction specify the index register you wanted of the eight, and that would get added before the instruction was executed. So you'd be able to pick up… if you want the 13th element of the array, you just put "13" in the index register.

Well, actually there was a little bit more than that. During the summer, I'd become a pretty good programmer. It really seemed very natural to me.

TH: So to write a program for this computer, you would need to punch it onto the paper tape.

WK: Oh, absolutely yes.

[10] The concept of the B-register certainly came from the group at Manchester but was not originated by Turing.
[11] He actually said "accumulator" but he misspoke.

TH: And then what happens next once you've got your program on a piece of paper tape?

WK: Well, you go into this room where there's this console, and on the right-hand side of the console there is this tape reader. There's also a Creed teleprinter and a cardpunch on the teleprinter. Behind you at the side, there are these cabinets. The room, about twice as big as this room, was full of cabinets, and the cabinets were full of vacuum tubes. I think there were 5,000 of them. Of course they created a lot of heat, so there was the rush of air, air-conditioned air rushing through the cabinets to cool the vacuum tubes. And when you got on the computer, it was yours for typically maximum of five minutes. You put your tape in there. I also built a tape winder so people could spool up, wind their tapes into these big spools. You'd put that into the reader and it would suck your tape in. Then it would execute your program. Then you would get a result, or maybe you wouldn't. And you'd get off the machine by the time your time was up, and the next person is right behind you, ready to start again, rebooting the machine from scratch to read the tape, and so on.

TH: And the output would also be paper tape?

WK: You could get it as paper, but it would typically be punched on paper tape so you could print it out at your leisure, so to speak.

TH: So then you'd take the paper tape over to the…

WK: You'd take it to another room and ran it through what amounted to a Creed teleprinter, and it would do what the teleprinters did with paper tapes. It would read the tape with little mechanical fingers that would poke up. If they hit a hole they would notice it, and if they didn't hit a hole it would stop, and that would set up a bunch of complicated little levers and switches to get the print head to work.

TH: And you have mentioned that doing this, you discovered that programming was something that you enjoyed and were good at.

WK: Oh, I got really good at it because… Well, all I can say is that the idea seemed perfectly obvious once I saw what was there. It was just the right thing to do. But this machine had a fatal flaw. Its mean free time between errors was about five minutes on a good day. On a bad day, it could be less. So I learned to write programs with redundancy and checkpoints and backup and all sorts of other stuff. I think I and Harvey Gellman were the only ones who were good at that. Harvey Gellman[12] worked for Atomic Energy of Canada and he used the machine for the atomic energy calculations. He and I were the only ones who could get the machine to do anything worth doing on a bad day.

[12] Harvey S. Gellman (1924–2003) Did his PhD at the University of Toronto and then began a successful career as a consultant. He was remembered as being "one of Canada's computer pioneers and most distinguished consultants".

Or on a bad time of day, because typically around five o'clock give or take, the machine would get very cranky. It was because industry was shutting down and they say housewives were turning on, and so there were a lot of electrical transients. Those electrical transients shouldn't have gotten through because the machine was powered by a generator that generated 400 Hz power, 400 Hz because that meant that the transformers needed for the filament voltage for the tubes could be much smaller. That generator was driven through a rubber-coupled flywheel by a motor that was driven from the power line. So the inertia of the flywheel should have absorbed any transients. And somehow the transients would get through, and we never really found out how.

Although we got some idea. I remember one… It was a summer's day. It was August. It was very hot. Windows were open. And Pat Hume, Professor of Physics and a friend, Pat Hume[13] and I were using the computer, and every now and then we'd hear a "clackety-clack" from down the hall and the B lines would fill up. So we'd reboot the computer and then there'd be this "clackety-clack" from down the hall and the B lines would fill up again. And Pat walked down the hall and I would tell him when the B lines are filling up. You know, shouting out the door. He found that the "clackety-clack" was one of these electromechanical desktop calculators, a Marchant, and every time the guy using it pushed the divide key, the calculator would go "clackety-clack, clackety-clack, clackety-clack," because division is a "try it and see if you like it" business, and if you don't, you back off and move the carriage. So now we knew what was doing the "clackety-clack," and that was correlated with filling the B lines. And what Pat did was ask the guy, "Look, would you unplug the calculator from this socket and plug it into that one? Let's see what happens." Oh, then the "clackety-clack" occurred but the B lines didn't fill up. And we still didn't figure out why that happened, because everything was in conduit. All the wiring was conduit, shielded, and so on. It was a mystery to the day that the computer died.

TH: And I think the biggest job that you tackled on that I understand was a simulated airline reservation system.

WK: Oh yes, that's right. Joe Kates was aware that I was a good programmer. I had written programs for General Electric's transformer people. I had psyched out one of the really devious codes that Joe Kates had written for binary-decimal and decimal-binary conversion. So he employed me in the summer of '54, starting early May '54, to write a simulation of an airline reservation system. The purpose was to show that if you can computerize the reservation system, you can cut down on communications costs which land on telegraph wires or telephone wires. Either way expensive, because people would call in from distant places saying, "I have a customer and he wants so many seats on such-and-such a flight," and "Is that okay?" and then they'd have to go through a huge rolodex system. It was very elaborate.

---

[13] Pat Hume has been called a pioneer of Canadian computing. He was a professor at the University of Toronto where, in 1952, he began his work developing software for Canada's first electronic computer, FERUT.

I actually spent almost the whole month of May studying exactly what they did, watching their operation, watching the codes they use, watching the two-letter codes for the airports – at that time they were two letters, now three. I came, I thought, to understand really well exactly what they[14] were doing and why they were doing it. Then it was time to write up flowcharts so that I can get the computer to simulate this. So now what you'd do would be to enter your request to the computer and instead of having to wait for somebody in a distant city, you'd get the response in two seconds. The computer would maintain the inventory of seats on the airlines and so on.

The only problem that really weighed on me was the fact that the mean free time between errors of five minutes was death to a demonstration. The businesspeople who would come to see this demonstration could not tolerate a machine that crapped out every now and then. So I had to put in all of these measures in order to reduce the part of the program that was vulnerable to crash to as few instructions as possible. All the rest was done redundantly, backed up on the drum, and the drum[15] was checked redundantly, and so on.

And the flowcharts were getting pretty elaborate by the end of July when Joe Kates said, "How's the program coming?" I was showing him a flowchart and he says, "Well, where's the code?" I said, "Don't worry about that. The code will write itself. We have to get the flowcharts right." So he got a friend of his, a physicist, Joe Shapiro to come and help me. Then with Joe's help, the translation took about two weeks and then we had the program running. Alright, so maybe it took another day or two to debug it, but the program was running, and now what were we going to do with the time?

So first what we did was prepare manuals. These were designed to look exactly like the manuals that the TCA people would use to learn about what are the operations that they can request and what codes do they send on the teletype machine to say, "I want reservation or I want to cancel or this or that." After I'd studied that, we ensured that the Creed teleprinter at our console could be used just like their machine so that if you typed in the same stuff, it would go to the computer and it would execute the simulation of the same operation on the inventory. Then we still had time left over, so we put in management things like cancel flights, launch a new flight. If there's going to be the Stanley Cup for example in Edmonton, you get an awful lot of people flying to Edmonton, so they'd lay on extra flights. Sometimes they'd cancel flights if bad weather or low demand.

Then we put in some very limited ability to suggest alternate routes. Because although the Trans-Canada Air Lines route looked like a fish spine that was running more or less parallel with the border, and then there were little spines that went up to places like Ottawa and Calgary and down to New York and Chicago and I think Seattle – I'm not sure about Seattle – there was a certain amount of redundancy in the route. So we made up a table which said, "If somebody wants to go from hither to yon but the direct flight is booked, why doesn't he take a flight that goes from hither to an intermediate spot to yon?" and we would suggest that.

It worked beautifully. At the end of August, all these guys in gray suits came, and looking over our shoulders, some of them looking over one of their shoulders as we type this stuff, said, "Oh,

[14] Trans Canada Airlines, now Air Canada
[15] He actually called it a "disk" but later corrected it to "drum"

let's see if you can cancel this flight." Of course there was no such flight. The computer said, "There is no such flight." Or "Let's see if I can get 13 seats on that flight," and the computer would say, "Well, 13 is too many." And so on. It worked just the way their system worked.

And you would think that they should be delighted, but they weren't. You see Joe and I had explained the system and we were there at the demonstration. I was wearing a red sports jacket. It was really sort of tweed, red tweed sports jacket. And Joe was wearing a green tweed sports jacket, but they were all in gray. They were looking at these two young sprouts who were going to take over their life's work. And they wouldn't do it. So Trans-Canada Air Lines shelved this thing and I went on, got married in September, and became a graduate student.

I went to England for two years. When I came back in 1960, Trans-Canada Air Lines had finally retired these guys and now they were going to go and set up a reservation system, but I think the company that they chose… I don't remember its name. I think it was something "Packard[16]." The company that they chose didn't get it done, and ultimately Trans-Canada Air Lines bought the same system that American Airlines was using, the SABRE system. Then of course these systems grew. I mean nowadays they have everything, not just your name and so on but how to get in touch with you and what have you been doing in the past and so on. Well, that was a big job.

TH: If Trans-Canada Air Lines had been interested in building on that to a real system, do you think you would have stayed with that project?

WK: Well, the first thing to do is get reliable hardware, and that would have changed things a lot. Also, by the time… How shall I put it? This was 1954. By 1957, machines were already in existence with memories very much larger. By 1960, larger again. So that and the ability to interact, the I/O channels for example on an IBM machine… An I/O channel was a little computer in its own right, and it had direct memory access. So the whole architecture would have to be thought out over again. It would have been an interesting problem, but I had even more interesting problems I thought.

TH: So at that point, you already knew that you wanted to go to graduate school?

WK: Oh, uh… Well, I know I wanted to work with computers, and where would I find a computer? I could move to the United States, but Toronto had the computer and I had the problems. So I was working on my master's degree and discovered, "Something interesting is happening here. It's not just a master's degree. I'm able to reproduce results that don't require the hypotheses that people used to think they needed." And…

TH: And that was a master's degree in mathematics?

WK: This was mathematics, that's right. So the hard part was the proof. The tedious part was running example after example after example to find out that what was going on was predictable. Well, the rate of convergence was predictable within a factor of 2, which considering that we

[16] Ferranti Packard – the Canadian division, formed by the merger of Ferranti Electric and Packard Electric, of the English firm of Ferranti Ltd

were going to increase the rate of convergence by an order of magnitude by choosing the right overrelaxation parameter, a factor of 2 was really inconsequential in the rate of convergence.

So I managed to find some papers, I think it was Stein and Rosenberg, a paper[17] in one of the London math journals, which triggered a realization that what they had done could be adapted to bigger and better things. So I was able to prove that this behaviour that I had observed was predictable and was very much like the behaviour that you'd expect if this combinatorial property was present. So you didn't need the combinatorial property. It meant that instead of having square meshes, you could have hexagonal meshes or irregular meshes. As long as it was a diffusion problem. A diffusion problem has a certain reciprocity which says that "The influence of this node on that is the same as the influence of that node on this." Roughly speaking, that's what it says.

But I'd also figured out a lot of other things that were interesting. For example, I wrote a program for Professor Coxeter that figured out from generating relationships what is the smallest group that can be generated that way that uses something called a Todd-Coxeter coset enumeration algorithm. Of course he was delighted because now he could deal with groups that were very much bigger than he could have handled by hand, but you know they weren't very big. They were just bigger than he could handle by hand. Other than that, nothing particularly awesome about them.

So it seemed like a good idea to continue this. I remember a fellow graduate student, Peter Bandler – B-A-N-D-L-E-R – Peter Bandler and I both wanted to work on this Todd-Coxeter scheme, so we flipped a coin and I lost. So I ended up working on these diffusion problems. Peter never finished. And it's not surprising, because in order to get real traction, you had to be clever. I think it was John Leech[18] in Edinburgh, he wrote what was in the late '50s certainly the best such algorithm.

But I'm very lucky. I had all this computing experience. The engineers let me take the computer over on Friday night. Not to take it home – that's what your draft says.

TH: I think… Yes. A transcription error, I suspect.

WK: Yeah, I took it over from them on Friday evening and Sheila came down, and while my program was running on FERUT, she would babysit it and I could go and punch more tapes or print them out or whatever it took. If the computer crashed, it would usually crash because the memory

[17] P. Stein and R. Rosenberg, On the solution of linear simultaneous equations by iteration, J. London Math. Soc. 23:111-118 (1948).

[18] John Leech (1926 – 1992) was appointed as a lecturer in the Computing Laboratory of Glasgow University (Kahan was mistaken in saying Edinburgh) in 1959. In 1968 a new university was created in Stirling, 40 km from Glasgow. Leech was appointed as Reader and first Head of Computing Science at Stirling. Two years later he was promoted to a Personal Chair, the first awarded by the University of Stirling. He took early retirement in 1980, having worked part-time for a few years before this due to ill-health.

filled up, and so we had an idea where it would go. And what I did was to put a little loop there that it would beep. So Sheila would come and say, "The baby is crying," and I would go and reboot it and start it up again. And then if need be, I could wheel around a cart that had an oscilloscope, and if the computer malfunctioned, sometimes it would run a diagnostic program. Not everything worked but some things worked, and then I could see where the bug might be according to the diagnostic program. I could go to a cabinet and open it up, and I could touch one of the vacuum tubes and "Pssh!" and I knew it was too hot.

Well, that was interesting too. You see, they used 5,000 vacuum tubes, so they knew they were going to consume a lot of vacuum tubes. Where were they going to get them? Well, Second World War was over and there were warehouse full of vacuum tubes intended for radar sets on aircraft. Somebody had decided that the vacuum tubes would be lighter if instead of steel or glass envelopes they used aluminum for the envelopes. Well, aluminum is permeable to hydrogen, so the tubes would get gassy. If you could have looked inside, you would have seen that beside the red glow of the cathode, you would have seen this purple glow. It was hydrogen atoms bouncing. Once that happened, the vacuum tube was no longer a switch. It just wouldn't turn off, and then it would overheat. That was one way to find out which vacuum tube had to be replaced, and I was allowed to do that.

So I would keep the machine going in many cases. There were some situations where I really didn't know what to do. Just had to shut it down. The engineers would come the next morning and try to figure out what was going on. From Friday night till early Monday morning, I had the computer mostly to myself. If someone came in on the weekend and wanted to use it for a little while, that was okay. You had to be humane about these things.

TH: Now we start actually the ENIAC book[19] with a quote from Hartree, who had written one of the first public descriptions of ENIAC. And something of an understatement, but he was trying to give a sense of what the availability of electronic computing could mean for mathematics, and he said, "You can do quite a lot with 10 million multiplications." So did you have a sense as you were tackling these various problems that mathematics was entering a new kind of era in terms of what could be done mathematically?

WK: Oh, not just mathematics. It was just everything. In science and engineering. The biologists weren't interested at the time, but they've come around since.

In the late summer of 1954, Sheila's family had a party at which I was to be introduced to her family because we were going to get married in September. And one of her uncles was a very prominent lawyer in Toronto, Nathan Strauss, Queen's Counsel. Very prestigious. And he felt that he should, well, check out this young guy, and he said, "Look, I understand you're into computers." "Yes." "Well, what are you going to do when four or five computers have been built and that's all the computing that the country needs? What will you do with computers after

[19] Thomas Haigh, Mark Priestley, Crispin Rope, ENIAC in Action: Making and Remaking the Modern Computer, MIT Press, 2016

that?" And I said, "Oh, they'll become companions." See. That was really done more to pull his leg than anything else, but prescient in a way.

TH: Yes. Would it have been possible for you to have done the work that you did for your thesis without a computer? Could you have tackled that?

WK: I don't see how. It would take far too long, and I don't think the idea would have occurred to me that you didn't need this hypothesis because I wouldn't have had the energy to run one of the electromechanical calculators on all the different kinds of meshes that I tried. On the computer, it took a while to program the thing. It might have taken a couple of days or three to write and get the program onto tape and debug it. But then it would go in a flash by comparison. Alright, so it took a few hours. But that's instantaneous compared with what it would take if you had to do it on electromechanical machines. And I was in charge of electromechanical machines for quite a while.

TH: Now you got your PhD in 1958. Before that, I know you spent a summer at the University of Illinois where they had their own computer.

WK: That's right. Oh yes. Well, you see it was clear that FERUT was… Well, it was out of its depth as computers went. The thought was that Toronto and the University of Illinois might collaborate on a new computer that they were thinking about. David Muller – M-U-L-L-E-R without an "E" after the "U," Muller – he had an idea to use what we would now call self-timed or self-clocked circuits. These are logical circuits which send out a completion signal when their result has become stable. If you do it that way, you don't have to distribute a clock signal and synchronize everything, because when you synchronize everything, you got to allow for the slowest, which could be a lot slower than other things, so it slows the computer down all told.

But it became clear pretty soon, both to me and anyone else who would look, that this would require an awfully large number of these newfangled transistors, and each one of the transistors came in a little can like that[20], you see, and where were you going to put them all? We didn't have integrated circuits then. And now that we do have integrated circuits, the question is not where we'll put the transistors but where are you going to put the wires? So we don't have very many self-clocked circuits. There are some. There are some very clever divide circuit that use… I think the name of the guy at Stanford who did this, I think his name was Feller. I'm not sure. I'm an old man and my memory has holes in it.

But once it was clear that the ILLIAC II was not going to be born, Toronto went ahead and got an IBM 650 and later an IBM 7090. Meanwhile that summer, which was hell in some respects because it was 95 during the day, both temperature and humidity, and the only things that flourished were the cornfields and the bean fields, and the only places with air conditioning were the computer room and the student lounge. And the student lounge was already occupied by students on couches trying to get some sleep out of the heat. But alright, so I managed.

But I met some really interesting people. Gene Golub. He was my host for my first week or so there. After that, he went off to Bell Labs. But he and I became friends from then on.

[20] Here Kahan is showing the tip of his little finger.

And I met several other guys whom I found instructive. One of them was Bill Gear, C.W. Gear. Gear was reputed to be the only sane person around, and he was. I mean he was really very steady. But his thesis advisor was Abe Taub – T-A-U-B – and Taub had a personality peculiarity. If he detected a weakness, he would prey upon it. That taught me something important about what not to do with graduate students. Their life is uncertain enough and so you really don't want to shake them. But Taub and Gear would get into arguments, which could be heard through the closed door and down the hall. But what was interesting was that Gear gave as good as he got, and he did it with a slightly Cockney accent. He had cultivated a more plummy accent at college, but he could revert to a slightly Cockney accent. So he wouldn't take this junk from Taub. But alas, Gene Golub did and was I think damaged by Taub. Taub mellowed when he came to Berkeley. He and I were on reasonably good terms, but I don't think he had very many graduate students after that.

TH: Yes. And I should mention for people who are interested in pulling up those stories, I did oral history interviews for SIAM with both Gene Golub and Bill Gear, so those are also available[21].

WK: They corroborate the story, do they?

TH: Essentially I think yes. Now when you got your PhD in 1958, then you were able to spend two years at Cambridge in England.

WK: Yes, yes. That was because I wanted to work with Hartree[22]. Hartree had come through Toronto in I think '57, in autumn, late autumn of '57, and I thought what he was doing was marvellous. He was computing the energy levels of molecules. These could be revealed by their spectra, but what you really wanted to know was how can you arrange them in order to minimize the total energy involved in their oscillations. And that computation was terrific. That's what I wanted to do.

So we made the arrangements to go to Cambridge and I applied for a postdoc fellowship, which fortunately I got. Now this occurred after Kelly Gotlieb had taken me to Detroit, to a meeting where I had presented the results that became my thesis that I just told you about, in 15 slides in 15 minutes. Glass slides. They had to be prepared by a photographer. And it caused quite a stir, I guess. So I was what the mafia would call a "made man," and that may have helped get the postdoc fellowship support from the Canadian government.

[21] These interviews may be found at
http://www.computerhistory.org/collections/catalog/102746795 for Golub
http://www.computerhistory.org/collections/catalog/102746786 for Gear

[22] Douglas Rayner Hartree PhD, FRS (1897 – 1958) was an English mathematician and physicist most famous for the development of numerical analysis. He was closely associated with many of the early computing projects in Britain and America and was very influential in communicating ideas among them.

But Hartree died before I even set out. I'm sure his family was distraught, and it upset me too. But all the arrangements had been made, so we went to Cambridge. Well, it was really not such a bad thing. We had a marvellous time. I met lots of really interesting people, both at Cambridge and Oxford, and London. And Gene Golub turned up after a while. He was taking his postdoc there also. And if you can imagine packing Sheila and me and Gene Golub and our luggage into a Volkswagen Beetle that we drove and took a hovercraft ferry across the Channel and drove to Paris for a congress of what became the International Federation for Information Processing societies – that was in '59 – how did we ever fit all of us in that car? I don't think I could do it now.

Among the people I met was J.C.P. Miller[23]. Now he became my *pro forma* thesis advisor because, since my PhD was from Canada in the colonies, they didn't recognize it, and so I had to enrol there as if I were going to be a graduate student. Oh well. So I had to pay some fees. It was like contributions to a charitable organization.

I should say by the way that after I came home, I heard nothing from John's College until a few years ago when they decided that I should remember them in my will. So they've been sending me stuff ever since.

But anyway, J.C.P. Miller had been computing for decades on hand-cranked machines and also on a National Cash Register machine, which was sort of an accounting machine that produced difference tables automatically. And he was a devious old coot and he taught me all sorts of tricks which he thought couldn't be used in electronic computers anymore because you just can't see the digits, they go by too fast. But I got to use almost all of them to figure out what to do. So that was great.

Now Wilkes[24] was very hospitable. We did have a small clash because someone had to teach Hartree's numerical analysis class, and Wilkes thought he was up for the job and he wasn't. Wilkes was a competent and clever guy, but I think he thought numerical analysis was trivial, and it wasn't. He mangled some things and I remember being called into his office and being told, "Kahan, I can appreciate your questions better if you ask them privately rather than in class. After all," he said, "this is an elementary class." And I said, "Well, there is a difference between elementary and superficial." And his face darkened and that was the end of that interview. And strangely enough I've heard from friends that he tells that story about me, same story as I tell about him, each of us to illustrate something about the other guy's personality. But I have no complaints about Wilkes. He was hospitable and I enjoyed the two years in his Math Lab.

---

[23] Jeffrey Charles Percy Miller (1906 –1981) was an English mathematician and computing pioneer. He worked in number theory and on geometry. He was an early member of the Computing Laboratory of the University of Cambridge. He contributed various algorithms and procedures to the construction and documentation of mathematical tables.

[24] Maurice Wilkes (1913 – 2010, head of the Mathematical Laboratory in Cambridge, himself a Turing Award Recipient.

TH: I think the other future Turing Award winner that you met on that trip was James Wilkinson.

WK: Oh yes, oh yes. Well, you see I had started to figure out backward error analysis while I was still a graduate student, in somewhere around 1957-58. It helped me explain an awful lot about how computations actually worked. And I wasn't the only one. Wallace Givens at Argonne National Lab had a glimpse of this in 1954. Turing had a glimpse of it in 1948-49, and even mentioned it in a throwaway line, one of the last paragraphs of one of his papers. But it took us until the late '50s for the penny to drop. And Wilkinson caught on in Teddington near London, and Fritz Bauer caught on in Munich. We all caught on approximately simultaneously.

So when I went to Cambridge, I would travel down to Teddington to meet with Wilkinson and show him what I had just done. You know, "I've now figured this out for eigenvalue calculation or for this calculation," or whatever. And he would reach up on a high shelf and pull off an envelope and "Pff!" blow the dust off and pull out some pages, "Oh yes, I think I've got that here. And do you have this too?"

And, well, it was great. I felt I was with someone who was in many ways a kindred spirit, but he was really very good at explaining things lucidly. It would read like a novel. So it was a pleasure to read the stuff he wrote and I was happy that he wrote the stuff up and I didn't have to.

TH: Now to pull back slightly, so all the people that you've mentioned were pioneer users of electronic computers.

WK: Yes.

TH: Now clearly error analysis is something that predates electronic computing.

WK: Oh yes.

TH: But what was it about mathematical practice with the availability of extremely fast automatic computation that meant that existing methods were no longer sufficient?

WK: Okay. Well, the atmosphere was poisoned by von Neumann. He disparaged floating point. His explanation in his written reports… Abe Taub was editor of von Neumann's works and he allowed me to sit in his office and leaf through and even borrow things for a while. So I looked through very carefully and I found that the only reason that von Neumann gave for disparaging floating point was that a good physicist should understand the range of his variables and could do his own scaling and didn't need floating point to do it.

But I didn't find that out until afterwards. What I had found out was that numerical analysis was a somewhat crippled subject. Kelly Gotlieb had given a class on numerical analysis in 1956 and it wasn't very good. It was out of a text that was the latest text, but the text wasn't really very helpful. The problem was partly that engineers were using floating point on computers, including FERUT, despite what von Neumann said, because it just didn't make sense to try to do all the scaling yourself. You'd have to understand what all the intermediate quantities meant, and on electronic computers, you ended up with a lot of intermediate quantities because the computations got longer, much longer than you could afford to do by hand.

So von Neumann was mistaken, and his mistake influenced others who believed that floating-point arithmetic is refractory to error analysis. It's just too complicated. You never know what the range of the variables are, and so what do you mean by "the last bit fluctuates according to the exponent"? How do you take care of all that? And I figured out how. Not for every computation. There are lots of computations for which we still don't have error analyses. You run the computation and you pray. But the major computations were susceptible to this kind of error analysis.

And when I had to write a thesis, I had asked myself, "Well, which of the various things I've been doing should I write a thesis about?" and Griffith[25], my thesis advisor, said one day I guess it was in March of 1958, he said, "Kahan, write a thesis. I don't care which of the 10 theses you could write. Choose one and write it and have it in my hands by May, or find another thesis advisor." Well, I revered that man and so I took him very seriously, and Sheila and I went out and brought a typewriter with mathematical keys on it so we could type up my thesis, which then had to be limited to whatever there was on the typewriter.

I chose the mathematically most challenging thing, which I've told you about – this rate of convergence problem. Then in 1957, Kelly took me to this Wayne State thing in Detroit and I gave this talk and it made quite a stir, but at the same meeting, the seeds of my idea's demise were sown. Peaceman and Rachford[26] described an alternating direction scheme, which if you choose the parameters appropriately, convergence is even faster. And to this day we don't know exactly why. For rectangular regions, we understand it, but for arbitrary-shaped, like banana-shaped regions, we don't really understand why the thing works so well as it does. In any event, their scheme was subsequently overtaken by what we now call multigrid methods, which are very much faster again.

Well, if I had my thesis about backward error analysis, maybe it would have had a more lasting impression. I don't know. In any event, Wilkinson and I were kindred spirits about that. We saw things the same way. What's more, so did David Wheeler[27] at Cambridge.

Now David Wheeler was involved in the microcoding of the EDSAC 2. He was really a genius, but a very reticent genius. In order to get information out of him, it was… well, it was like trying to open a can without a can opener, really. But if he did say something, you really wanted to think about it. He had also understood… I don't think he had invented it, but he had understood backward error analysis also. So some of my ideas could be bounced off him.

[25] Byron Alexander Griffith, a Toronto academic influential in mathematical and early computer developments.

[26] Peaceman, D. W.; Rachford Jr., H. H. (1955), "The numerical solution of parabolic and elliptic differential equations", *Journal of the Society for Industrial and Applied Mathematics*, Vol. 3 (1), pp. 28–41.

[27] David Wheeler was first a graduate student and then a staff member at Cambridge University. He made many significant contributions to the development of the EDSAC computer, and is often credited with inventing the concept of the subroutine jump instruction.

There were other people at the Cambridge lab. Peter Swinnerton-Dyer, who was extraordinarily bright, really one of the brightest people I've ever seen. And my officemates were no slouches either. Colin Cryer[28], that's the name of a South African. It's not in those lists, but he was doing really interesting work on partial differential equations with moveable boundaries, boundaries that would move under the influence of the solution to the differential equation.

So we had a great time. In the meantime, I was working on various algorithms and insights, helped partly by J.C.P. Miller's tricks, in order to better understand how approximate computation could survive at not just a thousand operations per second but they were getting up to a million per second. You see, very soon, 1963, they were doing a million per second on the CDC 6600. So that pretty much determined my career. I was going to be an error analyst and figure out how floating-point computation worked, why it worked when it did, and why it didn't work when it didn't.

TH: Now with the machines you've been describing so far, am I correct that there's no floating point in hardware, so it would be then with subroutines?

WK: Well, that's right. On FERUT it was a subroutine family designed by Pat Hume. The thing about FERUT was that as early machines go, it had a very good subroutine management system designed I was told by Alan Turing. There was a part of the memory that was reserved for the subroutine calling sequence, and it would automatically overlay your program with a subroutine that you called, executed, and then bring your original program back off a drum in its original state but now with the results from the subroutine. But the IBM 650 had decimal floating point built in, and so had the 7090, but it was binary floating point built in.

TH: So let's get on track with the story then. In 1960, you returned to Toronto as a faculty member.

WK: That's right.

TH: Did they have already a 650 when you returned?

WK: No. By the time I returned in the… I think it was late August of 1960, they had already installed a 7090 to supplant the IBM 650. However, something funny was going on because people were running the 650 emulator on the 7090. Why would they do that? It slows the 7090 down dreadfully to pretend it's a 650.

Well, the reason appears to have been my matrix package. I wrote a matrix package for the IBM 650. It was very convenient. Apparently while I was off in Cambridge, unknown to me it was getting heavy use by statisticians. Really by statisticians' clients – that is to say the psychologists and the life science people and so on who use statistics. They had used the scheme on the 650 because the 650 had to be programmed by assembly language. There was something called "FORTRANSIT" – F-O-R-T-R-A-N-S-I-T – and it was a crude form of FORTRAN, but it was inefficient. If you really wanted to use the machine efficiently, you'd have to program it in assembly language, and programming matrix-handling stuff in assembly language is just no joy.

[28] Last I heard of Colin W. Cryer was that he was a professor at a German university (added after the interview)

But I had found a way to pretend that the instruction set of the IBM 650 had as its operands not decimal numbers in memory but matrices. So the add operation, if interpreted this way, would add two matrices and put the result in the standard place, and similar for the multiply operation. The divide essentially did matrix inversion or solve the linear systems of equations. I'm pretty sure I had Jacobi's method for eigenvalues in there, and just stuff for preparing matrices and stuff like that.

And I handed that package over on a Thursday so that there was time on Friday to pack for the ship that was going to sail on Sunday to take us to England. When I got to Cambridge, I let them have my phone number, so I said, "If you've got any trouble, call me," but they never called me. So I concluded that no one was using it. Oh well. Only to come back and discover yes, they'd been using it very heavily, so heavily that they couldn't wean themselves away from it when they got the 7090.

Well, as it happens I did write programs for the 7090, I did write matrix-handling programs. Matrix-handling programs including dealing with complex matrices, matrices consisting of complex numbers, which I got to go very, very fast, because instead of calling complex arithmetic subprograms to do the complex add, subtract, multiply, during matrix stuff, I did it all with real arithmetic. So it went very fast. No subroutine call overhead.

Apparently Ontario Hydro became addicted to my programs because they have to solve complex linear systems in order to understand the electric distribution networks. They have inductance and capacitance. They're really transmission lines. So when Toronto switched from a 7094 that ran this program of mine to the IBM 360, on the 360 it couldn't run the code. I forget exactly why. I'd have to think about it now. So the chemistry department took over the 7094 and put it in their basement and continued running it for several years, and Ontario Hydro ran their codes in the chemistry department's basement for several years until finally they couldn't get spare parts. And that was the end of the 7094. But it lasted for a long time. I'm sure they had it for at least five years after I left Toronto.

TH: Now picking back up with the floating-point story, so you've mentioned that the 7090 obviously was a much, much larger, more reliable machine than FERUT.

WK: Oh yes.

TH: But you'd mentioned one of the differences was that it had floating-point hardware support, and specifically for binary floating point.

WK: Yes. Correct.

TH: So what difference did the presence of floating point in hardware make for the kind of work that you were doing?

WK: Well, for one thing it went an awful lot faster, not just because the… Ah, the 7090 had I think it was a 2-microsecond cycle time to read from core memory, and it took maybe a hundred microseconds to do a floating-point multiply or maybe a divide. I don't remember which.

Multiply I think. But a hundred microseconds is a lot faster than 3 milliseconds. Then the 7094 was even faster, about twice as fast. It had the same instruction set with a few frills. It had double-position operations. But it could run the 7090 code faster. And it had what we would now call 128K bytes of memory, which at that time seemed enormous. So of course it meant that scientific and engineering computations that were hopeless in the '50s were now routine in the '60s.

TH: And what did that change in terms of error analysis?

WK: Well, for one thing, differential equation solving became more practical. Differential equation solving is sort of an inchworm process. I think I've described it to you. You know, you take small steps, hoping that if the steps are small enough, you'll follow the correct trajectory and not get too far off it. But these things were plagued with certain kinds of errors, combinations of instabilities, because of the unfortunate choice of a numerical method. Although Dahlquist[29] the Swede figured out in 1956 what it was that made predictor–corrector methods unstable, so we ended up with a better idea of which numerical methods to use. But still there were accumulations of errors just simply because every step had to get rounded, and that's partly why I came up with compensated summation, which I wrote a note about that I think it was 1960-61. But I wasn't the only one who did it. There was a guy in Denmark who figured it out about the same time. So that made it possible then to solve trajectory calculations, orbit calculations that would go on and on and on and on without being overwhelmed by rounding errors. There were many computations, especially eigenvalue computations, matrix computations, there were many of them where it turned out that a certain amount of error analysis would give you confidence that you understood why you were getting the result you were getting.

Now unfortunately, backward error analysis has been badly misunderstood. People describe it as follows – I've got a book where a guy actually says this – "I can't give you the answer to a problem you gave me. I can give you the answer to a slightly different problem. It's a pity you didn't give me the slightly different problem. That was your mistake." Well, that's a misunderstanding. It is as if there's an intrinsic limit to how accurately you can solve a problem because of backward error analysis. And if you don't get a good answer, well, backward error analysis explains that that's what happened and that's the justification for getting a bad answer. And not just for matrix calculations. For example, the logarithm of a number x near 1 is closely proportional to x minus 1 plus higher-order terms, powers of x minus 1. But if you believe that the argument to the logarithm function is uncertain in its last digit, well, then x minus 1 will largely cancel, leaving you with nothing much more than the last digit or so about which you don't know very much, and so you can't complain if the logarithm function is wrong.

That is a mistake. It's a *serious* mistake. It's a serious mistake because first of all backward error analysis when it's applicable… It doesn't always work. Some computations don't submit to backward error analysis. But when it works, it's an explanation, not an excuse. It says why things happen this way. So if in your computation the result you get is no worse than if you had perturbed the data by *n* figures but your data is uncertain by hundreds of units in the last place, not just a few, well, in that case the uncertainty in your answer is more because of your data than because of the algorithm. If on the other hand the uncertainty you get is intolerable, that you

[29] Germund Dhalquist (1925 – 2005), a renowned Swedish numerical analysist.

can't stand having perturbations in *n* figures, well, then carry more digits. Do it again with more digits. You don't just acquiesce. It's an explanation, not an excuse. In the case of the logarithm code, somebody says, "Look at my logarithm algorithm. It's no worse than if you perturbed x by a unit or so in the last place." Well, that may be, but for all I know, x is exactly what somebody wants. It isn't uncertain by a unit in its last place. So if what you do is screw it up that way, then you're giving me a bad answer.

And it *did* screw me up. The IBM logarithm did that on the 7090, and it caused one of my differential equation solvers to malfunction. It took me weeks to find out what was going wrong. And so I fixed the logarithm program. Actually I even made it a little bit faster.

TH: Now I understand you finished up making quite a few changes to the system software…

WK: Oh yes, indeed. Well, the system software, the math library is not now regarded as part of system software. It's regarded as part of the compiler nowadays. But yes, I rewrote the log and the exponential, I think sin and cos. Oh, square root, absolutely, because square root hadn't been monotonic. How can you have a square root that isn't monotonic? If you increase the argument, you expect the square root to… well, if it doesn't increase, it should at least stay the same, but not decrease. The graph of square root you see looks like this[30]. So after a while it bends over. Doesn't flatten exactly, but you can change the argument without changing the square root, but you don't want it to go back. And that screwed up one of my programs, so I had to rewrite that one.

And we contributed these things to SHARE[31]. A guy at the University of Chicago, Clemens Roothaan[32], asked, "Well, why don't you just rewrite the whole library?" I said, "Well, because I'm a professor. I have to meet classes." "Okay," he said, and he gave the job to one of his programmers, Hirondo Kuki. Kuki did a superb job. He rewrote the whole math library, not just the parts of it that I had written. That put the subject in a different light. It meant that you could analyze the errors in floating point, that you could get answers that were correct to within a unit or less than the last place, that you could do it. And if you could do it and it wouldn't hurt performance much, then you should do it. It was as simple as that.

Does that answer your question?

TH: Yes. I think it leads us to another aspect. Obviously implicit in what you're saying is that the application programmer would have a good understanding of what was happening, of how much accuracy was needed, and so on. And I know that you…

WK: Well, some did and some didn't.

TH: Yes. I think you had also mentioned that you tried to… you made some changes to the error messages …

[30] Kahn uses his hands to show how the graph of the function begins to level out

[31] SHARE was a volunteer-run user group for IBM mainframe computers that was founded in 1955 by Los Angeles-area users of the IBM 701 computer system.

[32] Clemens Roothaan was a professor of Chemistry with his own IBM 7090.

WK: Oh yes.

TH: …so it would be reported to the programmers to try and expose that information to them.

WK: Well, getting an error message in octal wasn't very helpful. The floating-point error messages came from something called a floating-point trap routine, FPT. What I did was rewrite it, not only to give better error messages but to give retrospective diagnostics. Retrospective diagnostics means that if you are willing to accept a patch of some sort… For example, if you say, "Well, if a number overflows, replace it by the biggest possible number. If it underflows, replace it by zero. Or actually put gradual underflow into the machine," because hardware supported it, believe it or not. So what I did was to store in the operating system a cell that said, "Here is a list of the exceptions that have occurred and nothing has been done about them other than to put in the default fix-up." And I supplied code that said, "If you want, you can clear this flag. But if you don't clear the flag, it will stick up." And then at the end of your job when you get the accounting record, it might say, "You had an unrequited overflow at such-and-such." Lovely word, "unrequited." And so on.

Well, I didn't get a lot of thanks for that, except once. A mathematician had thought that he had discovered a counterexample to a famous conjecture, but it turned out that his counterexample existed because division by zero in the 7090 gave you a quotient of zero unless you stopped the machine, and he didn't stop the machine and he didn't know that. But my message said, "You have an unrequited divide by zero at such-and-such a place." It hadn't occurred to him that he had a division by zero, and that came just in time to save him from sending in a paper to publish his counterexample.

TH: And that theme of floating point not just being hardware but the interaction between the hardware and software environment is something that, as I understand, runs through the remainder of your career.

WK: Oh yes. Well, they're married. So they have to learn to get along with each other.

TH: Now we should say I guess just a few things about the broader development of your career, first at Toronto until 1968…

WK: Say, could you speak up a bit, please?

TH: Sure. Sorry, yes. We should say a few things about your broader career, both at Toronto until 1968 and since then at Berkeley.

WK: Yes. Okay. Well, I told you I got back to Toronto in late August, end of August 1960. I thought I was going to work with the two numerical analysts. One was Jimmy Chung in the statistics department and the other was Boris Davison in the physics department. But Chung had died in August. He had some lung problems. I think he was living on one lung. And Boris Davison died in November. There I was, the only numerical analyst.

Now that wasn't necessarily good news because I was fairly demanding. I wanted students to understand what they were doing. But there are many students who take classes in order to get the credits, and there were quite a few like that and they would have fared badly in my classes. I don't think it's a good idea to scoff at students who may be slow. It's like poking fun at somebody who's got only one leg. But I do get exasperated when people are in class and they don't want to know. In fact, I sometimes get exasperated with Sheila if she doesn't want to know. Well, she doesn't like to know about mechanical things other than that "It'll work if you push this button or that button." Then of course if she breaks it, our contract says I have to fix it. So I was willing to deal with students who were slow on the uptake, but unwilling to deal with students who just wanted the credit and didn't want to actually do the work.

And there's a lot of work in numerical computation, because when I started numerical analysis at Toronto, the students did not have easy access to a computer. I mean the IBM 650 was fully subscribed with people who had serious computations. It wasn't used for educating undergraduates. So I used slide rules. Okay. So all of my problems were slide rule–accessible, which by the way takes a good deal of effort, to find interesting computations that are portents of things to come but you can do it with a slide rule. Anyway, that's what I did.

After a while, it was clear that I should not be unleashed on freshmen, although in that freshman class there were two outstanding students. One went into physics, became famous. Another one went into mathematics. She also became famous. In fact, she became for a while head of the Fields Institute in Toronto. Betty Keyfitz. Was that right? I think that's… Well, I'm really bad with names, especially women's names, but I'm pretty sure Keyfitz was the last name. She was very, very bright.

I really should digress a bit.

TH: Oh, we don't have much time for digressing.

WK: Well, this is important I think. At the time, there weren't very many women in the mathematical technologies. They were rare. They were rare in physics. One of the few women in physics was Harriet Thompson, who was in my class, and she was absolutely marvellous. She went on to get a PhD at Cambridge and got it shortly before I got mine, and then left with her husband to go to South Africa. And when I turned up in Cambridge, people would think first that I was American. "Oh, you're Canadian. Oh, then you must know Harriet Thompson."

Harriet Thompson really shook the place up. For one thing, she and her fellow graduate students would often go to places that had been restricted to men alone. She would come in and some flunky would say, "Oh, no, no, you're not allowed here," and she would ask, "Why not?" Then her companions would say, "She's with us." Oh, there are lots of things she did that made it… Let's say she was a pioneer that made it more respectable for women to go into such things as physics and so on.

And at Toronto, we needed such pioneers too. Among them, one who worked for me, she was assigned as a programmer to work for me. I think it was sort of a part-time job, summer job. Marion Green had graduated at the top of a math class at Toronto. I think her husband was second. He went into government work consulting on computing and she for the time being got this part-time job and was working for me. She was so bright, so quick. I remember that I had to sit down with her and have a serious conversation. I said, “Marion, you’ve got to go to graduate school and get a PhD, and then instead of me telling you what to do, you’re going to tell me what to do.” Well, I don’t know whether that frightened her or not. She decided ultimately that she would rather stay home with her babies.

Many women made that decision. We didn’t make it easy for them. It was very unfair. We lost a lot of really good talent that way. That’s important enough that I felt I had to say that.

TH: So what brought you to Berkeley?

WK: Ah. Beresford Parlett. He was a friend whom I had met in I guess it was 1964 in New York at some conference or other. He also worked with matrix computations and he was English. He invited me to come to Berkeley to help set up a computer science department. Now that was ’68. In ’66, I had spent half a year at Stanford with my family. And as we were heading for home, my younger son who was just about two and a half at the time said, “Daddy, why can’t you get a job in California?”

So that and Beresford’s invitation worked upon me and my family, and I thought, “Well, why don’t we go off to Berkeley for say seven or eight years?” Seven or eight years because it says in the Bible that after seven years, a slave is entitled to his manumission. He’s entitled to be released. And if he decides he doesn’t want to be released, then he is to be branded on the forehead and then becomes a slave for the rest of his days as a member of the family, slave member of the family. Anyway, that’s where the number seven and “sabbatical” comes from.

Sheila didn’t want to go. Sheila had a large network of friends in Toronto, friends from her college days. She had gotten a degree in philosophy in 1955 and then a graduate degree in social work in I think it was 1956 or ’7. She specialized in geriatric social work, which has come in handy now. She takes care of me. And she didn’t want to go, but I thought about it and I figured that she’d like it once she got there. And she has liked it. She has a wide network of friends, including quite a few widows.

So we came here and found a really strange situation. Not a computer science department. It was going to be one of three computer science departments, one of them clandestine. Lotfi Zadeh in electrical engineering was forming his computer science department as a condition, that he was allowed to do that. That was the condition he extracted from the administration so that he would assent to their setting up a computer science department in the School of Letters & Science. But there was also the Computing Center, and the Computing Center had some guys in there who were attracted because of the thought that ultimately they would become part a computer science department.

That didn’t quite happen that way. When I was brought to Berkeley, I had a triple appointment – math, computer science, and a little sinecure in the Computing Center. Well, that Computing Center thing had been started by Abe Taub. He had been brought in to run the Computing Center under the auspices of the math department. He was brought in specifically because the math department feared Lotfi Zadeh, who had a well-earned reputation as a shark. The math department thought, “We have to bring in a shark to offset Zadeh.” By the way, Zadeh is still alive, but he’s very old and sort of hunched over. Comes in from time to time.

Well, it turned out Zadeh was a better shark than Taub, and so Taub didn’t really manage to set up a mathematical enterprise at all. But he did enlist some people who thought that they were going to be part of a computer science department. It included a guy from the math department, René DeVogelaere. The Computing Center also employed a guy from electrical engineering, Professor Martin Graham. Martin Graham died just last year. And they had a couple others and me.

But they also had an annual deficit of $150,000, and no matter what they did, it seemed that this became a constant of nature. They couldn’t get rid of this deficit. To make the deficit worse, the federal government finally cottoned on to the idea that people were using computers financed by federal research grants also as teaching instruments to teach undergraduates how to program, and the federal government said, “No, you can’t do that. You can’t use our money for that purpose.” Well, that was a serious blow because computers at that time cost between $400 and $600 an hour no matter how fast they went, and teaching undergraduates consumed a lot of time.

So as they tried to get funds from various places, they discovered they’d always had this deficit and a committee was formed to see how the deficit could be relieved. I was a member of that committee. And I agreed with the…

TH: What year was this?

WK: Beg your pardon.

TH: What year was the…?

WK: ’69 I think.

TH: So not long after you arrived?

WK: That’s right. I arrived in January ’69 and I think by September of ’69, I was on this committee. With a couple of other guys, a guy from chemistry and one from I think demography was it? Can’t remember. I can’t remember.

Anyway, what we figured out was part of the deficit was caused by this sinecure. There were people in the Computing Center who were supposed to give advice that was hardly ever wanted. So we recommended that the sinecures be terminated, that the Computing Center find a competent director. It was currently being directed by a sort of temporary director because the previous director had become unpopular. He had just made the deficit worse. So the administration said, “Yes, we’ll accept your recommendations,” and what they did was promote the acting director to director. He was incompetent. They eliminated the sinecures except for Marty Graham who consulted on communications networks for the campus. I didn’t need the sinecure because I had a tenured position in math and one in computer science. René

DeVogelaere was a little bit annoyed because the math department didn't want to take him back for a while, but ultimately they were compelled to. And a couple of other guys had to be let go. I think I earned their undying enmity for my recommendation that this thing be squashed. And you know what? The next year, the deficit was $150,000 again.

TH: Hmm. So that got rid of one of the three computer science departments you have mentioned. Were the other two eventually merged?

WK: Well, what happened with the L&S computer science department was that we were promised a certain number of FTEs – full-time equivalents. You know, we could appoint a certain number of people. But then demonstrations began around 1970-71. Students wanted an ethnic studies department. Finally the administration knuckled under and they compelled the Dean of Letters & Science to set up an ethnic studies department, and he only had a certain number of FTEs and he had to take one or two from us.

Well, what happened then was that we wanted to appoint a very bright woman who got her PhD at Stanford in programming languages, and the dean was looking for any excuse to renege on his promise. And he got word that one of the members of our department and this young woman were tight, that this could therefore be considered a nepotism case. So they used that as an excuse to deny her appointment, which was really rather annoying.

The chairwoman of the statistics department, Elizabeth[33]… What was her name? Oh gosh. Her surname has fallen into a hole in my memory. Maybe it will come back in an hour or so. She got wind of this and she observed that the relationship between this young woman and one of my colleagues was only a rumour. They didn't have actual witnesses. So she said that they had denied her due process and so on, and she was going to make quite a fuss. So ultimately we did get her. She was appointed.

But that was the last appointment that we could make. We couldn't grow. So ultimately it appeared reasonable to merge our computer science department with Electrical Engineering and Computer Science department. This wasn't a bad thing for all of us. Marty Graham had been in EE and he had become chairman of this computer science department when the merger occurred. And it didn't bother me because my hobby was electronics. It was a forced marriage, but it wasn't a really bad thing. And ultimately we ended up, all the computer science people but a few ended up in the new building, Soda Hall, just on the north side of Hearst, just across from the campus. Some stayed in Cory Hall, which was just across the street on the south side of Hearst. That was electrical engineering's domain. I think the department has flourished, and I think it's partly because rather than despite the forced marriage.

TH: Now numerical analysis was a much more prominent part of computer science in the '60s when these disciplines first came together.

WK: Oh yes.

TH: So how did that change over the course of your career?

[33] Perhaps it was Elizabeth Scott, I am not really sure (added after the interview)

WK: Well, computer science[34] has become what I describe as "sliver under the fingernail" of computer science, because although numerical computation is more, there's more of it and it's more important than ever before, and especially floating-point computation on a grand scale… For example, biological systems are being simulated right down to the level of mitochondria inside the cell. You set up a bunch of differential equations, exchange equations, and delay differential equations, and gosh, that's floating point in vast quantities. And the astronomers are into much more of it now that they've been discovering exoplanets and they wonder how big they are, and physicists of course have all sorts of interesting theories they're trying to explore, and chemists are still trying to figure out how to put various molecules together, bigger and bigger proteins, and other things.

So there's more and more of this scientific and engineering computation than there has ever been, but there's a lot more of the other stuff. Games, entertainment. Games use floating point a lot. They're simulating the movement of these avatars and other strange figures. They've got to figure out what you can see and what you can't see, how do things move, if you drop something it splashes. There has to be, well, a certain amount of artistic verisimilitude I guess you'd call it. It doesn't have to be a perfect simulation, but has to look more or less like what you'd expect to see at least in a cartoon. And there's music videos, there's communications on a vast scale. Neural nets are now big consumers of floating point, but it's very short floating-point numbers. I don't think anybody knows what a neural net is doing, which I guess is why they're using it.

But the niceties of numerical computation are generally appreciated only by a small number of people who take graduate classes in numerical analysis, with one or two exceptions. One exception is the graduate class on big matrix computations. It's a math class, it's a graduate math class. Always heavily subscribed. It's taught by a former student of mine, Jim Demmel, now a colleague. He also teaches a CS class on parallel computation that is also very heavily subscribed. The people who work in graphics, they have a numerical analyst amongst them who specializes in meshes, and meshes are used for regenerating surface information.

But the language people have other fish to fry, really. They aren't terribly interested in supporting numerical computation. Although some may say they are, but they soon find out that there are other things that are in greater demand. The operating systems people have never been particularly interested in numerical analysis, but there are lots of operating systems challenges, especially in memory management issues, a lot of problems with memory hierarchies, what's going on in the cloud, and what have you got in the cloud up there, what is it? And I could go on. The human interface people don't have a lot of demand for numerical stuff, and usually if they do, it's very simple.

So computer science has proliferated. It's almost as if it's the Greater East Asia Co-Prosperity Sphere, if you remember that.

TH: Japanese.

[34] Here he meant to say "numerical analysis" and not "computer science"

WK: Yeah, an enormous empire. That's what computer science is becoming. And it frightens some people who feel as if whatever they were doing before is now being taken over by the computer nerds. And I try to persuade my colleagues, "For God's sake, don't frighten the customers."

TH: Yes. So in the second half of the interview, we'll be looking specifically at your work as a consultant to HP and Intel, the IEEE standard, and the Turing Award. Is there anything else that you think you should particularly mention about your career at Berkeley? So you've talked about the origins of the department, you've talked a little about the changing role of numerical analysis.

WK: Well, I've taught numerical analysis classes, but I've also taught classical analysis, complex variables, linear algebra. I introduced a discrete math class. I'm not really good at it, but I thought it was really important, chose a good text which continued to be used for a few years. I felt that combinatorics should be a compulsory part of the syllabus for a math major. It has become that now, although I think they promised to me one year and then they reneged. It took a few years after that to finally get it going.

I helped students learn to solve problems ostensibly practicing for the Putnam exam, but I specifically said, "You don't have to take the Putnam exam to take this class and you don't have to take this class to take the Putnam exam. But if you want to learn about how to solve problems, I'll try to help you learn that."

Now my teaching style is very different from that of the folks who've taken it over. Well, it's been taken over by a couple of Russians, and they teach it as a class where they teach techniques. Now I don't do that because when you give students lessons on problem solving, you could use a book like Polya's book *How to Solve It*, a delightful little book, and there are others also on how to solve problems. What they do is they tell students certain principles that the students should follow, and of course students dutifully memorize those principles. But memorizing a principle is not the same thing as recognizing when it's applicable. What I would much rather do is have them struggle with various problems, I'd try to give some easy ones, every week some easier ones and some really hard ones, because there are a few guys who are such hot shots that the hard ones are the ones they need. And I try to… tried, when I was doing it, to cover the range of problems – algebra, geometry, a little bit of combinatorics, and problems off the wall.

Here's a problem off the wall if you want to hear one. Look, there are four ghostly galleons sailing on the sea so beset by fog that you can barely see from one side of the ship to the other. Each galleon has a steady course, steady in direction and speed, in order to reach its destination. One of the ships strikes another one amidships, and the captain of the ship struck amidships hears the captain of the other ship say, "Damn it, that's the second collision we've had this night." Then it happens again – the same ship is struck amidships. Now of course they're ghostly galleons, so they just pass through each other. They don't have any damage, you see. And it's the same – the guy on this galleon struck amidships hears the other one say, "Damnation, that's the second collision we've had this night." What must this captain of the ship struck twice do in order to avoid being struck a third time?

Well, I can explain it to you later. But you understand what I mean by an "off the wall" problem?

TH: Yes.

WK: Okay. Actually this is a problem in vector geometry. I got lots of them. I would scour The American Math Monthly and other places for a variety of problems, even theorems. Sometimes somebody would prove an interesting theorem. I would look at that and "You know, that would make an interesting problem for my class." I must have at least a couple hundred of them buried somewhere in some disk file with the solutions. The solutions are examples of exposition, mathematical exposition. My students who aren't accustomed to that… Look, some of them are inarticulate. They may have been born into an English language home but they're still inarticulate. Some are not so hot at English because that's not their mother tongue. And one of the things they need is not just to solve the problem but to explain the solution to persuade someone that "Yes, this is a solution."

That was what I wanted them to practice and I did this for over a dozen years. And, well, we didn't often win anything much. There was one year when we had a team come in… I think it came in second. Beat Stanford. But we're competing against outfits that can cherry-pick as we cannot. We cannot pick which undergraduates will be admitted to the math major, and therefore I can't pick which students are going to want to come and learn about problem solving. Furthermore – this is an unfortunate and important issue – some of the best students are so jealous of their grade point average that they will not take a course where they cannot see their way clear to getting an A, at least an A.

Then there's problems with women. They don't come! I had, I think it was in 2005, I had a woman, a Romanian woman who had won European Olympiads, gotten her PhD in New York, came to Berkeley. It was sort of a postdoc. I had her help me with the Putnam exams one year and I made sure that she figured prominently, and she did play a prominent role. I thought that would attract women. Didn't. It attracted more boys, men, because she was very pretty. Then a year later or so, I took a sabbatical and left it in the hands of this woman and another, a Russian, who was a Morrey Professor – she's now permanently on the faculty – and I left it to them to set things up and to do it their way. And I thought, "Maybe this will attract women." No. Both of them are very pretty, so it attracted more boys.

I don't know what to do about that. And it's not that women aren't bright enough. They're more than bright enough! They just don't believe it.

Well, you asked me about something important. I've told you a couple things that I think are important.

TH: Okay. Great. Well, let's break for lunch there and then resume.

WK: Okay.

After a lunch break the interview continued with part 2

TH: So having reviewed the earlier part of your career and talked in general about your time as a faculty member in Berkeley and Toronto, the second half of the interview will be looking specifically at your involvement in a number of high-profile projects. The first of these would be your consulting career particularly with respect to Hewlett-Packard and its calculators, and the work on floating point for the Intel processor chips. So chronologically, I believe it would make sense to talk about Hewlett-Packard first.

WK: Alright. I got into Hewlett-Packard because there was an advertisement by Texas Instruments of their calculator which made Hewlett-Packard's HP-45 look bad. The exercise was the following. "Type in your telephone number including area code. That's a 10-digit number. Take its logarithm. Take its exponential. Do you get your telephone number back?" Well, on the Texas Instruments machine you did and on the HP machine you didn't. And the guys at HP were worried about this. It cast what they regarded as unfortunate aspersions upon the integrity of their machine, their HP-45. I think it was Dick Sites who told them that I knew enough about floating point I might be able to help them with that. I can't remember who got them to get me because all I got was an invitation from somebody at HP to come down.

And I did and I showed them something interesting. I said, "Well, here's the TI machine. Look, I put in the telephone number for a telephone number in North Carolina. 919 area code. Logarithm, exponential. Oh, there it is back again. Logarithm, exponential. There it is, back again." And the seventh time, "Logarithm, exponential. Ah, the telephone number changed. What does that tell you is going on? Apparently there are more digits to the right of the 10 than you can see here." That's what was going on. The TI calculator had sloppy 13-digit arithmetic, which seemed to be better than HP's more careful 10-digit arithmetic, and their arithmetic wasn't as careful as it could be. But this was an intrinsic problem. You're going to get into this type of problem no matter how many digits you carry. It's just a matter of looking at all the digits in order to see that somehow you are able to reproduce log and exponential. Okay.

TH: Was this 1974?

WK: I believe so. "However," I said, "there is a way to get better results and to get rid of anomalies, and that is carry 13 digits internally for every operation. You'll supply 10-digit operands, but you do the arithmetic to 13 and then you round it back to 10, and you'll find that that has the least anomalous behaviour," whereas on the TI calculator I could show them all sorts of strange things. Even though you're only looking at 10 digits, I could show them some very strange things were happening, and that's because the TI calculator didn't carry what we call a guard digit. It wasn't very careful about it's transcendental functions, and so you'd have anomalous behaviour.

On a programmable machine, the anomalous behaviour can be really, really bad news because the programmer has to assume something about the arithmetic, particularly when it comes to branches. If his assumptions about the arithmetic, based perhaps on intuition, maybe on an inadequate education, but if his assumptions are wrong, his program may go wrong. So I really wanted them to improve the quality of the arithmetic, starting with the trig functions and transcendentals like log and exponential.

And they did it. Dennis Harms, he was the chief mathematician in the calculator group. He'd gotten a PhD in math at Cedar Rapids I think. So he implemented this. It didn't take him very long, and he demonstrated that yes, the arithmetic did seem to be much cleaner. Yes. They showed me a calculator. I think it was the HP-27, a little shirt-pocket machine about so big. I saw that yes, the trig functions and everything behaved better, but I said, "But what are these buttons here? N, I, PV, FV, and PMT. Five top-row keys. What do they do?" "Oh, you don't have to worry about that." So I worried about it and I discovered that they were the financial keys. Then I thought about it and I said, "How can they do that? That's not easy." It's not easy to compute the interest rate, it's not easy to compute things that involve what amounts to exponentials and logarithms and so on in order to get nice, tidy formulas.

So I found an instance where instead of delivering 10 figures they were delivering only about six. And I phoned up my contact at HP and he said, "Well, six will be enough." I said, "Yeah, but if I have an example where you just get six, there will be another example where you won't get that many," and I found one where now you had only two or three digits. So now he got worried. Then I said, "And here's what you do in order to avoid this problem," and I showed him that now that they were carrying 13 digits in some other computations, they could exploit that in order to make the financial calculations really clean.

So I showed him the code and he tried it, and the next thing I knew I was invited to a meeting of Hewlett-Packard people including David Packard. The problem was that they had a warehouse full of these HP-27s, and now that they knew that there were defects, what should they do? Well, they could send them out and tell people, "If you don't like it, send it back and we'll fix it or replace it," or we can just simply take all the ones in the warehouse, unsolder the ROM and put in a new ROM. And to David Packard's credit, he said, "We're not going to sell something which we know has a repairable defect." That and other comments made me think extremely highly of him. He seemed… Well, our word for it in Yiddish is a *mensch*, a real person.

So they fixed it and then the HP-27 was able to do financial calculations as reliably as any of the other things. But that led to work on a bigger financial calculator. It's sort of a desk machine like this[35] with a big display. That was the HP-92.

TH: Let me just ask you a general question before you talk about that. So to what extent would these anomalies that you've talked about be connected with the feature that generally computers are holding internally numbers in binary but that the arithmetic that humans are concerned with is decimal?

WK: I think you've got to put your question again because I'm not sure I understand it.

TH: Sure. I mean you've spoken about a couple of specific anomalies and you've talked about a number more, and I'm sure you could list many, many, many, like a defect here, a defect there, or a workaround, a way to get around it. Now ENIAC and machine calculators had decimal arithmetic in the hardware. Humans are concerned with numbers in base 10 primarily, but the internal hardware is doing everything in binary. So are a significant proportions of these

[35] Here he is illustrating something about the size of a large loaf of bread with his hands.

problems exacerbated or caused by the difference between the binary arithmetic and the decimal?

WK: Yes. There is a serious problem for someone who is unaware or has forgotten that a machine's arithmetic is binary, or hexadecimal for that matter, but what he sees displayed and printed is in decimal. So what you see is not quite what you've got, and when you try to put a number in, unless it's a moderate-size integer or a few others, what you put in is not what gets there. Not quite. And that causes distress on occasion because you may know what the correct result would be if the number you put in the machine was the one that was really there, or if the one you saw being acted on by the machine… Because you display an intermediate value, but of course it's displayed in decimal. It's really in binary, and it goes in and something else comes out? Weird.

This is prevalent in spreadsheets and Microsoft for many years, and still for all I know, has tried to cover up the fact that the Excel spreadsheet is doing binary arithmetic though everything is displayed in decimal. That is a serious mistake because whatever way you choose to cover up this fact to make it appear as if it's decimal will simply exacerbate the anomalies later. They'll become quite inexplicable. And I've written about that on my webpage. Errors designed not to be corrected.

I believe that decimal arithmetic is more humane. We have, you and I, already discussed some of that. The new IEEE standard – new as of 2008 – merged binary and decimal arithmetic together so that many of the rules and so on apply to both, but the decimal arithmetic was not what you think. It was a very subtle concoction. We can go into that later. I believe decimal arithmetic is what ought to supplant binary arithmetic for all casual computations and many that aren't so casual, because what you see is what you get. It's going to be easier for people to understand what's going on. Even if what's going on is anomalous, at least they can see the numbers and they really are the numbers instead of phantoms of some sort.

For example, MATLAB prints numbers that look like integers. If the numbers are spaced close together, they really are integers. If they're spaced far apart, then they aren't the integers you think. They're just floating-point numbers that happen to be awfully close to the integers and they got rounded to those integers. That's a fairly subtle distinction, you see. I mean, how often would you notice it? And it can be very annoying too, because again what you see is not what you've got, and sometimes the difference matters.

However, I think that for the largest large-scale computations in science and engineering, people won't really care if it's binary. That will be the least of their worries.

TH: Okay. So, sorry, returning to the Hewlett-Packard machines, so you mentioned then they were working on a large desk-sized calculator.

WK: They were working on I think it was an HP-9000 machine that was about so big[36]. It was a programmable decimal machine with a reasonable amount of memory, and I was lent one of those in order to develop some of the algorithms that were used later. I think my work with HP, which continued when they moved the calculator division from Cupertino up to Corvallis,

[36] Here he uses his hands to illustrate something about 2 feet by 18 inches

Oregon, that work was very fruitful. I enjoyed the people with whom I was working. And there were some interesting stories.

One interesting story arose because the subdivision of the calculator division, the subdivision with whom I worked which made small calculators, little handheld shirt-pocket calculators, was told that they had incurred enormous losses up there in Corvallis. Enormous losses, but their calculators were the ones that were selling, and the other guys were behind or they're on the fab line, they don't sell directly, and so they aren't making any money. "We're the ones who are making money. How could we be the ones incurring enormous losses?"

Well, after a while the accountants figured it out. You see, they were charging every subdivision *pro rata* to their sales. So if the other subdivisions didn't have any sales, that meant that these guys in the small calculator division were bearing all the expenses including the fab lines and development of this machine and that one. Then of course they realized that these guys were the ones who were keeping the lights on.

So when I went up there one day, they told me that this is what happened, that they were going to have a celebration. They were having a picnic outside. They have a spacious lawn there. It was lovely green. Beautiful day. And I thought I'd bother the project manager, Stan Mintz, and I wanted to get a "solve" key on the calculator. I wanted to arrange that if it's a programmable calculator and you can program the left-hand side of an equation, that then I'd like the calculator to search for the root, and I had an algorithm for doing it which I thought would converge at an adequate speed and a way of deciding when to quit and so on. That's what I wanted to put into a calculator. Stan had resisted me because he'd said, "I've asked this of the marketing people, and they say no one has ever asked for a 'solve' key." Of course they wouldn't ask for a "solve" key. It wouldn't have dawned on them perhaps that they could have it.

But finally we were at this picnic and Stan spiked the punch, which is very much contrary to HP rules. I'm teetotal, so it didn't affect me, but Stan got jolly. He said, "I tell you what, Professor Kahan. I'll let you have your 'solve' key if you can provide me with an 'integrate' key. I was never able to integrate when I went to college. Give me an 'integrate' key and you can have a 'solve' key too." Well, how could I resist that challenge?

So I did come up with an "integrate" key that would fit in the space. The space available was very small. Just a handful of registers were available for scratch. And Dennis Harms helped me work on it and microcode it. So we got it working, and then the "solve" key, and they got put into the HP-34C. And I have one, but its battery is dead so I'm not going to bring it out.

Well, the thing was though that the HP-34C's "solve" key and "integrate" keys were now interesting challenges since things could go wrong. Some equations don't have solutions. Some equations give you the wrong solution because when you compute the left-hand side of the equation, roundoff gives you junk. And integrals can be even more interesting. So I wanted the guys who write the manuals to include this kind of information in the manual so it would help people use these very powerful keys correctly. But HP had a policy, and the policy was "We don't put tutorial material in our manuals because we're dealing with professionals like us. We know what we're doing, they know what they're doing. And besides, Professor Kahan, are you telling us that you've persuaded us to put something in a calculator that can get wrong answers?"

Well, I seduced the marketing guys and they put in an extra chapter in the HP-34C manual, and then their manager said, "Hey, I told you not to put that in. Take it out." And they said, "Well, if we take it out, we're going to delay the completion of the manual, and that will delay getting the calculator on the market." So there he was, blackmailed in effect. He had to leave the stuff in, and they won a prize at the Willamette Valley technical writers' association for technical manuals, partly because of these chapters. And when the salespeople did surveys, they found that several of their customers said, "I bought the calculator because those chapters tell me how to use it and I can use it more easily than I can on mainframes." So we were vindicated.

In the meantime, the financial calculators were getting complicated, and that's because of something called internal rate of return. I've got one of the financial calculators here. Yes. It has an "IRR" key. Now internal rate of return is a fictional interest rate, but explains a transaction where you've got a certain number of cash flows. Say initial cash flow goes into a bank and then you get some cash flows out, and maybe you'll have to put more cash in sometime, and finally you get all the cash out and you exhaust the account. And the question is, when you look at all these cash flows back and forth, what fictional interest rate would explain these sums of money? Because what you really intend to do is not put the money in the bank, you intend to invest it in some project, in some product. It will cost you so much to develop the product, then you'll start selling them. Oh, and then there's going to be taxes down there of some sort, and maybe that's going to be an outflow. So once you've got this schedule for the expenditures, which may be the first 10 cash flows, the cash going out to build it, and then there's an income and you expect to sell it for a certain period of time, and now what we want to know is, well, what rate of interest are we making on our investment?

That is the internal rate of return. It requires that you solve an equation which can be fairly complicated. In fact, under some circumstances, you could have a polynomial with degree in the tens of thousands to solve. But the calculator only does only so many arithmetic operations per second, the slow ones are seven arithmetic operations per second, and you'd really like to get the answer in 20 or 30 seconds because most people won't wait any longer. That was a challenge, and I found a way to do it.

Actually it's written up in my lecture notes on real roots of equations. But at that time, people hadn't thought about that. That made a lot of the difference. Now we had a calculator, financial calculator. Not only was it accurate, the error's confined to one unit in the last digit displayed. Not only is it accurate, but it solves the internal rate of return problem. And it solves these problems, including some from mortgages and, oh, various kinds of investments, discounts, it solves them accurately and quickly, whereas on the then-competing Texas Instruments' Business Analyst, you could put in the data and the calculator was supposed to solve a problem, but it would go catatonic, or it would run, run, run, run, run, never stop, or it would give you a junk answer. And that wouldn't happen on my program.

So I recall I thought I'd proved it. So we really had a superior beast. That was the HP-38C. That was the first one. Then this guy[37], the HP12-C. And having produced these and found them very well adopted and people really liked them.

What HP liked best of all was that every now and then they'd get a phone call from someone who'd say, "Look, I've got this calculator and it's giving me an answer different from the one in the tables." "Oh dear," the telephone answerer would say, "We'll have to look into that. Can I have your number and we'll get back to you?" So what they would do would be to run the same calculation on their big HP 3000 and check as best they could, and they'd always find that the calculator was right, the tables were wrong. After a while, they had a list of tables that were wrong. After a while, the telephone answerer would say, "Oh yes, we know about that one. That table is mistaken. The number should be such-and-such instead of whatever you see in there." And that seemed to invite a great deal of confidence in this little machine.

So here I've got… This one here is a prototype. This is one of the… Oh, this is a prototype. It's one of the originals. It survived a certain amount of testing and then they gave it to me. In a box, I've got the 25th anniversary version. And here is the 30th anniversary version. That was in 2012. Where do you see an appliance, an electric or an electronic appliance on the market for 30 years? And so far as I know, it's still on the market.

Well, the success of that calculator made my next demand sound more credible. I wanted a calculator that had not just the "solve" and "integrate" keys. I wanted to have complex variables and small matrices, so that it would have everything for sophomore engineering students except divs, grads, and curls. Divs, grads, and curls require a larger display. Can't do it on that little window. The fact that you can do matrices on that little window, that's a little bit remarkable to begin with, but of course I represent matrices simply by their name – it's matrix A, B, C, D, or E. And if you want to know what the elements are, then there's a "walkthrough" key, which allows you to walk through it and see what the elements are, and the same thing for putting the elements in. But normally you don't want to look at the elements in the matrix. Normally you just want to do the matrix algebra until you get a final result and then you want to walk through it.

And I got them to build it. That was this one here. That's the 15C. It really is my favourite. Now in order to get them to build it, I had to make an estimate of a market. So what I did was go to our library here, and I looked up all the colleges I could think of that had engineering schools, and I looked up their enrolments, and I figured that about half of the incoming class each year would want to buy this calculator. Why? Because when I used the HP-34C, the one with just "solve" and "integrate," I would plant myself in the library or I would plant myself out in Sproul Plaza at a table and I would pretend to be working with this calculator. Students would come up and ask what I was doing, and if they were engineering students, they'd say, "Oh, and what does that do?" and I'd show them. And half of them would go to the student bookstore just on the other side of the plaza and buy one with their own money, not waiting for a doting aunt or someone like that to buy one for them. So I figured half the students are going to buy one.

TH: And would they have cost several hundred dollars?

[37] Here he refers to the calculator he was holding when he talked about the IRR (internal rate of return) key

WK: At the time, it would have cost about $150, but the price came down rapidly. It came down to rather less than a hundred. I think it was $70. Well, I think the financial one costs $70, or it's priced at $70.

Anyway, the marketing people wouldn't believe me. They figured that they could sell only a third of the number that I quoted on the basis of their experience with calculators. So they told Dennis Harms… Now Dennis Harms was in charge of mechanizing the production line for this range of calculators. Their little shirt-pocket calculators are put together by robots. Their subassemblies are tested by robots. There's 20 seconds of hand labour between the stuff coming in on the input dock, the various parts and so on, and the cartons going out on the going-out dock. Twenty seconds of hand labour on average per calculator. The rest is all done by robots. And what Dennis did was set up the production line so it would produce about half again as many as these guys wanted, which turned out to be about half of what I wanted, and they were selling by word of mouth alone. There has never been an advertisement in a Western language for the HP-15C, but they were always sold out. There were waiting lists for them. I think there was an advertisement in Japanese, but not in Danish, German, English, French, Spanish, none of those.

Well, I think I have to say that there should not be an expectation that these things would survive all that long, because after a while you get the tablets, the laptops, the PCs and so on, and it's not impossible to reverse engineer these things so that you can have all the benefits of the calculator and more on a big screen. So I didn't expect the things to last forever. But the 12C seems still to be soldiering on. I don't quite understand it. And about three years ago, or maybe a bit more, there was a demand for… I think it was 150,000 people asked on email for the 15C to be reissued. I think what they did do was issue 15,000 of them and they all got snapped up. And I was looking on eBay for something else and I came across an advertisement, someone wants to sell a 15C for $500.

Well, I'm not proud of a price, but I was proud of the calculator. MIT I think in one year if not two arranged to have one shipped in to be purchased by every incoming engineering student. And it satisfied my desire, my fervent desire to allow professors to offer problems to students that were at the same time more realistic and less tedious. I thought that would make for a better educational experience. Well, now of course they can do it on PCs. But now they can look up the answer on the web. So problems don't serve the same role as they used to.

TH: So your role as a consultant to HP was from 197-…

WK: I'm sorry, I can't hear you.

TH: According to our notes, so you were with Hewlett-Packard consulting from 1974 to 1984.

WK: Yeah, that sounds about right.

TH: And that would overlap with the work as a consultant to Intel, which you began in 1976.

WK: Yes, I think that's right. Now that had an interesting origin also. While I was at Stanford in 1966, I gave some classes on floating-point computation including error analysis and a little bit about how it was done. One of the students there was a graduate student actually, Gene Golub's graduate student, John Palmer. He got his PhD there but then went to work for Intel. And when they were developing the 8086, 8088, and the Intel 432 if I remember the number correctly, they wanted to get floating point on these things, and I guess Palmer persuaded them that perhaps it would be an idea for every box with "Intel" on the outside to have the same floating point on the inside.

He remembered the classes I'd given and so he brought me in and said he'd like to have a really nice floating-point design. I said, "Well, if you mimic IBM's floating point, you're going to have a very big market." "No," he said, "we want a good floating point." The story about IBM's floating point is interesting too, but that's from the 1960s, late '60s. "Alright," I said, "Why not copy the DEC VAX floating point? As binary floating points go, that's among the best." And he said, "We want the very best." "For whom?" He said, "For a mass market." "How big a mass market?" He couldn't say, but he wanted me to understand that this was for a really, really big market. That's what they had in mind.

I had to think about this, because you see, up to that time, if a computer was used a lot for scientific and engineering computation, it normally had a numerical analyst attached, very much in the spirit of the Asian elephants where they tie a little string around the elephant's leg and tie it to a stake in the ground. The elephant could pull it all out easily but doesn't. You can think of the numerical analyst as attached to the computer in the same way. But if they were going to have a mass market, there aren't that many numerical analysts. So I had to think about what would work not only for the specialist technicians, the numerical experts, but also for this mass market.

And that is what induced me to design the 8087 arithmetic the way I did. Now I had to do it with a stack because they didn't have opcodes in abundance. So the stack architecture was a little bit of a hindrance, but I got them to include the ability to reach down into the stack for an operand to combine it with a top, and that made it a lot easier. I had lots of experience with that kind of stack and I knew that it would function well.

TH: Now we maybe should just introduce the concept of a coprocessor.

WK: Say again.

TH: We should maybe just introduce the concept of a coprocessor at this point, since…

WK: The coprocessor?

TH: …yes, not everyone may be familiar with that.

WK: Ah, right. Well, of course nowadays floating point is on chip with the main processor. In fact, you might even get one per, umm…

TH: Core.

WK: …core. Thank you. But at that time, the density of transistors made it seem miraculous that you could get the CPU on the chip at all. So if you wanted floating point, that would have to be a separate computing engine, and you'd plug it into a socket near the CPU and there would be conventions which would say, "If you see an instruction of a certain kind, you'll know that it's a floating-point instruction instead of an ordinary instruction. And since the coprocessor and the main processor are both reading the same instruction stream, the coprocessor knows when it's time to execute one of its own instructions. However, the address computations have to be done by the main processor, and so the main processor has to squirt the final address over to the coprocessor. So there has to be enough wires between them."

TH: Yes. And of course minicomputers would have processors spread over many boards.

WK: Say again.

TH: Minicomputers would have processors spread over many different boards.

WK: Oh yes.

TH: With microprocessors, they could… As you say, managing to squeeze the whole of the processor on a chip even only with integer arithmetic was a real accomplishment.

WK: Well, even on the minis, on the PDP-11 for example, yes, the processor took up a board or two, but the floating point was also on a board. You could buy that board. And in order to exercise the board, you had to tell the operating system to activate it and also to say whether you wanted to do float four bytes wide or a double eight bytes wide. This was a nuisance because an operating system call would take longer than the floating-point operation. So Kernighan and Ritchie decided that they would do everything in double, save the time for conversation. And if your variable was a float, that's okay. You just pad it with zero so it becomes a double. And if you want to store something as a double, you just throw away the bottom half and store the leading half. That turned out to be advantageous, strangely enough. They used to get better results doing arithmetic on data that was all floats. But it was geometrical computations, some of which are a little bit dicey, especially for games and scenery. And they got better results than they would have gotten on the mainframes of the time working on the same data.

But then the C language took off and CDC and Cray, they got annoyed because their single-precision was as wide, very nearly as wide as everyone else's double, and their double had to be done in software and it was slow. So they asked for permission to do FORTRAN-ish evaluation. If you got an expression all on floats, you just evaluate everything in floats, whereas Kernighan and Ritchie would have said, "If you've got an expression with floating-point variables, it will all be evaluated in double. Whether the float's double, it doesn't matter. In fact, even integers, if they got mixed in with floats, then you'd get evaluated in double."

So there was a standards committee. I guess it's for… I can't remember the organization. J-something-11. It wasn't the IEEE. I think it was the ISO. But there was a standards committee that said, “Okay, in C, you can use FORTRAN-ish evaluation,” and then everybody did because they could use the same rear end for C as for FORTRAN. And then these beautiful results went away.

TH: Back to the 808-…

WK: HP.

TH: …the 8087.

WK: Or 8087?

TH: Yes. So I just looked. The 8086 itself apparently had 20,000 active transistors, and I understand you had 40,000 in…

WK: Well, there were 20,000 in the ROM, and what’s more, of those 20,000, the Israelis figured out how to pack two bits per transistor instead of just one by using very, very artful and delicate sense circuitry. So they managed to get all the microcode in, not only for add, subtract, multiply, divide, square root, assistance for decimal-binary conversion, and log essentially – it was really log(1+x) , exponential, the exp(x) – 1, tan, and arctan. And these kernels, as I call them, would make it easy for software to compute all the other transcendental functions – sin, cos, hyperbolic functions, inverse hyperbolic in arcsinh, arccos, and so on. You could have a math library.

And there was a math library written by a friend of mine at Intel which was pretty good. Of course some compiler writers insisted on using their own library, much to the detriment of reliability and repeatability. But the library was going to serve the third floating-point format. The 8087 chip could bring in operands that were integers or four-byte floating-point numbers, or eight-byte, or 10-byte. And the stack operated on floating-point numbers 10 bytes wide with 64 significant bits. This was more than floated double, but it was based on longstanding knowledge. They used to have a scratchpad with extra digits so that for every operation that you would like to appear atomic, you can implement it in software in the scratchpad round it back to the working precision and out it goes and it looks beautiful.

So for example in MATLAB, ideally when you multiply two matrices together, you’d like to accumulate the scalar products for each element with extra precision. Then you’d round it back to the ordinary precision and you would get a much better matrix product, especially in the circumstances where otherwise a lot of cancellation would occur.

That was the design and I said to John, “It’s 10 bytes wide now only because when I asked your engineers about the longest carry propagation they could tolerate within a clock cycle, they said, ‘67 bits.’ Well, we need a guard, round, and sticky, and that leaves 64.” And that’s what determined the width of the word. “But,” I said, “if demand grows for wider precision, this can grow to a 12-byte word and 16-byte word. All you want to do is make sure the programmers are aware that when you’re programming with this extra width” – we called it an extended format – “all you can rely upon is that you have at least a certain number of significant bits, at least a certain exponent range. And if you really want to know how many bits you’ve got, you do what we call environmental inquiry" – which was very easy on this machine – "and it tells you how many significant bits you've got. Later if they want to put in a chip with a full 16-byte-wide quadruple-precision word, that's okay because your environmental enquiry will tell you that's how many bits you've got, 113."

So I thought that this would be a design that would last into the future, and it would make it possible to implement math libraries extremely reliably. You had these extra digits. Like on the HP calculator, you got a 13-digit scratchpad in order to generate digits, of which 10 are actually going to be displayed and stored. The same principle applied here.

TH: So basically what you're saying is that the chip would not only allow enormous speed-up compared to doing things in software, but also basically by building best practices into the hardware, it would make the software much simpler and more accurate.

WK: Simpler and more accurate, yes. You could afford to use an algorithm that might lose eight bits. But that's okay, you've got 11 extra bits. Lose 8? Who cares! When you finally deliver the result, it will be a nice double-precision result, correct within a fraction of a unit in the last place.

TH: Yes. So you're not just designing a piece of hardware. You're also very much bearing in mind exactly how it would be programmed.

WK: Yes. And I insisted upon flags. That meant that if an exception occurred, either the hardware would not trap it, give you a default result, but it would raise a flag, and so you could test the flag at your convenience. You see, in Java where you're not allowed to test flags, if you want to know that something bad has happened, you've got to test after every potentially aberrant arithmetic operation. Who's got the patience for that? It turns your code into spaghetti. But if you've got the flags, you can speculate. "Let's do this operation. There's a chance of one in a million that it won't work. At the end I'll test the flag, and if it didn't work, okay, then I'll say, 'Sorry about that. We'll go back up here, but now we're going to do the job in a different way, which I hope will be more robust but it's somewhat slower. Sorry about that.'"

And this is a sane way to do it. It corresponds to throw-catch-finally, but without the interference of the operating system. It means everything can be handled in the math library, and that includes the possibility that if you divide zero by zero, you don't want to get a NaN[38]. Maybe you know that this occurs only because the correct result would be 13 if you took limits. Okay. What I want you to do is to be able to presubstitute and tell the system, "In this block of code, if you encounter a zero over zero, call it 13." Or call it "Q." We'll store the address of Q instead of the NaN, and when somebody gets a zero over zero – it doesn't happen very often – then a trap will occur. It won't go to the operating system; it will go to the math library. The math library will look into a table – "Oh, what does he want? Oh, he's given me this address. Okay, I'll fetch from that address and that's the result I'll give him. Okay."

TH: So the advantage of that would be the operating system really is not going to know what to do with it and the math library would?

[38] A special binary code which indicated that this was "not a number"

WK: Exactly. Unless of course the programmer has made no provision, and it might be that he enables traps and the traps have nowhere to go but up into the operating system. Well, that's bad. But programmers can do that anyway and I can't stop them, and won't. But now they have a better option.

Also, the flag should ideally serve for retrospective diagnostics, and that means once the flag is raised, it should point by hashing to a place – now this would be maintained by the operating system if it's requested – to a place in your program keyed to the source code so you know, "In my source code at line so-and-so, this event occurred." Wow! That gives you an *enormous* advantage in trying to figure out what went wrong. And maybe it didn't go wrong. Maybe it's okay. But that says you can look there and say, "Oh, is that what I did? Oh, that's okay. That'll work."

TH: That also would require the… well, I suppose the hardware, the math library, and the operating system, the compiler, and the programmer all to work together in a particular kind of way.

WK: Yes. Well, wouldn't it be nice if they collaborated? But you see our programs are getting hideously complicated. Scientific and engineering computations have burst the bounds of the most insane ambition of 20 years ago. And so now they're so big and so complicated that we can't debug them. I've written… It's called "Boulder.pdf". Boulder, Colorado is where I presented this, so "Boulder" – with a capital "B" – ".pdf" on my webpage. It explains what I think we have to do in order that programs be… well, floating-point programs be debuggable.

Now look, parallel programming has its own long list of ways for things to go wrong, and floating point isn't going to make that any better. But the floating-point exceptions are peculiar because there's so little you can do about them. They're very local. All you have to do to cope with floating-point exception is look to see "What were the operands? What was the operation? Where is the destination?" Then you go to the math library with that information and the math library can have things pre-set, like what to presubstitute. Or it can have something that says, "I'm going to generate a message." Or it can say, "Well, I'm going to abandon that computation. Let's jump somewhere else" – that's the throw-catch-finally.

All these options are available to you, but the most important one is that you don't have to test frequently, because those tests are murder. You see, computers look ahead in the instruction stream. They have an instruction pipeline and what they do is they first disassemble the instruction, the opcode, the address field and so on. Then they figure out, "Is this really the address we want or do I have to modify it and upgrade it?" And so on and so forth until finally they've got to the point where they know what they want to do and they know what they want to do it with, but it's taken several pipeline stages you see and they're ready to go at an issue rate for instructions which is essentially one per cycle. But if you've got branches, you can't do that, because you've got to have some idea of where you're going to go. After you branch, you're going to have to abandon the things that you've deciphered. And there's such a thing as speculative execution – "Let's go down both branches, and then at some we'll figure out that we can abandon what we did on one of these branches."

Well, that requires extra resources that could otherwise have been used for concurrency. I don't have to go into detail to tell you that tests and branches are not only going to rob you of performance. They're going to turn your code into spaghetti, and it's very, very hard to read and to understand. It's much better to have a statement at the beginning of a block which says, "In this ambience, understand these conventions. Understand that zero over zero makes 13, or infinity minus infinity makes zero." Or whatever it is you want. Then as you read the code, you can check as a mathematician, "Oh, what happens in the special cases?" instead of having to branch somewhere to find out what happens in the special cases, in which case you may have a compounding of special cases and things can get very hairy.

So that's what I wanted, and I didn't get it. I almost got it. I came pretty close, but no cigar. I'm still fighting with the language community. They seem… I guess the only word I can think of is refractory to this. They'd really rather do something else with their time. Because you see, you said it, there has to be collaboration. It's not just the programmer. The programming language has to afford these capabilities. It has to be tied in with the debugger so they can exercise appropriate options. It means that the math library has to be ready to handle things that otherwise the operating system would have had to handle. And the operating system has to let the math library handle it instead of insisting that every trap is going to go into the operating system because they're worried who knows what insecurity you might create.

TH: And lots of what you've said I think applies also to the floating-point standard that we'll be getting to. So let's backtrack ourselves in time. We've been talking about the 8087. Now clearly squeezing…

WK: Well, the 8087 morphed into the 387, which morphed into the 486, and then came Pentium.

TH: Yes, and the 486 integrated it onto the main processor.

WK: You know, it was a crock. You could buy a 486DX that had floating on it. You could buy a 486 that didn't have floating point. Actually it did, but it was disabled. Then if you had a 486 with a floating point disabled, you can buy a 486-something-or-other that had floating point on it, except it didn't – all it did was enable it on the main chip. Talk about marketing. Well, I don't want to spout a dirty word.

TH: So we said the complexity…, on the one hand squeezing all that onto the coprocessor required enormous ingenuity, but on the other hand the coprocessor was still approximately twice as complicated as the main processor. What kind of uptake did the coprocessors find? How widely instituted…?

WK: Oh, that's another terrific story. You see, Intel struck a devil's deal with Microsoft. The deal was that Microsoft would support this coprocessor, the 8087, in I think it was five languages. Let me see if I remember them. They were BASIC, Pascal, C, FORTRAN, COBOL. And Microsoft said, "Okay, we'll do it. But you, Intel, must not sell your own compilers in competition with ours," and Intel agreed to that. But Microsoft reneged. They did produce a

BASIC and FORTRAN and C. I don't remember a Pascal and I know they didn't do a COBOL[39]. Furthermore, they didn't really support the coprocessor.

I know that because I went up with a delegation from Intel to Redmond in 1982, and what we wanted to do was to help them understand what they needed in their compilers to support the coprocessor. And starting at nine o'clock in the morning, there were the Intel bunch and me on one side of the table, and there were the Microsoft bunch on the other side of the table, and we were compatible. You know, we were talking to guys like us. Everything was going along swimmingly until a boss came in. Bill Gates comes into the room, and, you know, "Hello," etc., and says, "Look, here's this PC on the table. You see that socket for the 8087? Almost none of them are going to be filled," he said, "and so it really isn't worth going too far out of our way to support it." Ah, you can imagine the cloud that cast over us.

What Microsoft did then was the following. Microsoft had its own floating-point software before this. The way you got it was the way you call any other subprograms – you push the operands on the system stack, call this subprogram, subprogram operates on these things, leaves the result on the top of the system stack and backs up. What they were going to do was to change their subprogram so that instead of the subprogram doing it in software they would simply invoke the chip, which means they had to then transfer the operands – remember they've got the addresses on the software side – they have to transfer the operands by fetching the operands for this address in some little buffer area and then moving it onto the chip. You then do the operation and then you go and reverse the process. You never actually use most of the registers on the chip.

That's what they did initially and it crippled the chip. It meant that instead of getting a speed-up by factors that could be as big as a hundred from the chip, you're getting speed-up by maybe factors as big as four. That was the worst of it of course, because you know performance is everything. But also they didn't support all three formats. They only supported the floats and the doubles, not the extended format. So it meant that the programmers in various languages didn't have access to a scratchpad where they could get extra precision in case they needed it.

Well, Borland did it differently. Borland produced a spreadsheet called Quattro, and the arithmetic for that was done by Roger Schlafly, the son of Phyllis Schlafly. Phyllis Schlafly's a well-known conservative nut. Roger wasn't always proud of her here in Berkeley, but anyway, he was the one who worked for Borland and he said, "I want to be able to use the widest format on a spreadsheet, because it's got almost 19 significant decimal digits' worth of precision and on the spreadsheet that's a good thing to have." So Borland's C supported all three formats and Roger wrote the Quattro code in Borland's C.

Which was just lovely except for one thing. Somebody figured out how Quattro could mimic Lotus 1-2-3, mimic it so well that if you had these macros for Lotus 1-2-3, the same macros could be made to work in Quattro. And Lotus sued. They said, "That's infringing on our copyrights." Initially the judge ruled in their favour. Borland appealed. Then Lotus got bought up by IBM, which was ominous because IBM can afford infinitely many lawyers. But ultimately

[39] Microsoft did actually produce a compiler for both Pascal and COBOL, but he is correct when he says that the languages didn't properly support the coprocessor

Borland won. The judge ruled that if Lotus really had wanted to protect these things, they should have been patented, if they were patentable at all. But copyright wasn't the way to go. But it was a pyrrhic victory for Borland because they had devoted so many resources to fighting this lawsuit that they fell behind in compilers.

Well, in the meantime GNU and others put in perfunctory or reluctant support for the third format, and ultimately Microsoft did the same thing. They didn't allow you to declare the 10-byte format as a variable type, but they used the stack and the 10-byte variable format in order to do the arithmetic. But now they killed one of the important capabilities of the chip, which was the invalid operation flag. If you do a divide zero by zero or infinity minus infinity or square root of a negative number in a real context, it will set the invalid flag. But also if stack overflow occurs, it'll set the invalid flag. Stack overflow, remember, was handled alas improperly because the Israelis and the guys in Santa Clara didn't really get together properly.

So stack overflow was a real pain to handle. It was best avoided. But Microsoft compiler wasn't designed to avoid it. They figured out that you'd have to have a fairly expensive expression to overflow, so what the hell. What they did was arrange that if at execution time the stack overflow occurred, this would be this invalid operation and Microsoft would issue this message at runtime, "Your expression is too complicated. Please simplify it." At runtime. Of course you might not have been the one who wrote the program, you're just running it, but you may then have executed a branch that the programmer forgot to test. And with this branch, it turns out the expression is too complicated. Programmer didn't know that. Now you've got this message. What are *you* supposed to do with it?

Well, that wasn't the only thing that was wrong. You see, it meant that Microsoft had to clear the invalid flag, because the hardware has the property that if the invalid flag is up and you enable the invalid trap, it will trap right then and there, and they didn't want that. They wanted the invalid flag to be down, the trap to be enabled, and if the stack overflow occurred, then you'd get the trap and you'd get this runtime message. But that meant that now I couldn't use the invalid flag to detect zero over zero or things like that, because the invalid flag would always be down. They always put it down. So that meant that you lost this capability.

Well, then of course Java doesn't like flags at all because they're side effects. So people haven't really gotten the benefit of it. Then even in places where there is support, it's thought that the flags were these hardware things, but they're not. The flags are really storage cells which may contain a pointer into a hash table so you can figure out what went wrong. But whenever you access this flag, what you do is you check the hardware to see whether a flag has been raised since last time you looked, because that would override the fact that you don't see a flag raised here – "There's one over there. Uh oh, got raised since I last looked." And that has to be understood, but it isn't understood. So people think the flag is just a hardware thing, and now that means that's it's a global variable. And global variables are a pain in the ass.

So what can I say? I guess I'd have to quote a military maxim whose origin I do not know, but it goes like this. "In every army large enough, there is always somebody who doesn't get the message or gets it wrong or forgets it." And that's what happened to some of these features of the IEEE standard and the coprocessor chips.

TH: So let's move back and talk about the standard now then. And I know a lot of the ideas that finished up in the standard I understand are coming from your work with…

WK: Yes. As soon as I heard that a standard committee was deliberating these things, then I attended one of their meetings.

TH: What year was that?

WK: I think it was 1977. I'm not 100% sure.

TH: Alright. So at that point, you had starting consulting for Intel, but still the chip itself had not yet appeared?

WK: No, no. But I have to warn you, I'm 82 years old and every now and then, something falls into a hole in my memory or gets bent. So if I say it was 1977 but it was really 1976, I just can't remember exactly. I think it was 1977 that I went to the first meeting. Al Despain[40] sicced me onto them. And I realized that there were all these people that had different floating-point designs, none of which was fully satisfactory for a mass market. Even the DEC design, which was a good design for its era, it really wasn't designed for a mass market. Not if you're going to have what could be, I thought, hundreds of thousands of people writing programs with floating-point in them. Hundreds of thousands. Millions! Millions and millions! But anyway…

TH: So there's two distinct problems then. One is that there's a lack of consistency between different architectures…

WK: Oh yes.

TH: …to the extent that if you run the same FORTRAN code, you'll get different answers.

WK: Sometimes drastically different answers, that's right. And Robert Stewart was worried about that because he saw microprocessors proliferating, and on each different company's microprocessors, there was either software or firmware for doing floating point and they all did it differently. So he set this thing going and I thought I had the only design that was really suitable for what appeared to be an enormous mass market, reaching down to people who had never taken a numerical analysis class and might be very clever about whatever it was they were doing, except they're not experienced with floating-point arithmetic.

So I persuaded Intel to let me divulge the rational and the algebraic operations – add, subtract, multiply, divide, square root, plus one or two other little things that don't matter. Intel agreed. Intel felt at that time, first, that they can do anything, and second, that they're leading the world, and part of leading the world is sharing just this kind of information. But what was still to be kept very, very secret was the fact that this chip was also going to include decimal-binary conversion assist and the kernel for the elementary transcendental functions. I think some Intel executive blabbed to an executive from a competing company at a golf game or something,

[40] Then a professor at Berkeley, later at the University of Southern California.

because it was generally understood that Intel was going to put an awful lot on this chip, much, much more than just these few arithmetic operations, and one could suspect that maybe this IEEE committee was formed to slow Intel down. Well, that's perhaps an unworthy estimate of the competition, but there were times when it felt that way.

Anyway, Intel agreed, and so the KCS proposal was born. "C" for Jerome Coonen, who was a graduate student. He had made the mistake of coming into my office and asking if I could explain what the PDP-11 was doing. One thing led to another and the next thing you knew, he was ensnared. Stone, "S." Stone, he was an experienced computer architect from the east coast, and his experience included drafting things for standards. So he helped shape the format for the document.[41]

TH: And you were the "K"?

WK: I was the "K," that's right. As such documents go, it was comparatively readable and it wasn't terribly long. So we put that forward, and of course there was resistance. The resistance came initially from people who said, "Hey, you can't do that. How can you round correctly without an extra carry propagation?" On and on. "And so you're going to have terrible performance penalties."

Well, part of the purpose of the meetings was educational. It was to show people that "Yes, you can do it without a significant performance penalty, and here's how." And transistors are cheap, so you use transistors in a certain way and gee, you could get it all correctly rounded. All it needed was guard, round, and sticky bit. It's three extra bits. People knew about guard and round, but that hadn't been so well informed about sticky, although there were guys at Michigan who knew about sticky way back in the '60s. So we showed people how to do it, including square root, how to get a correctly rounded square root. Well, it will cost you maybe one extra division over getting a square root that would otherwise have been regarded as adequate previously.

And as people learned that they could do it, they became… the engineers in any event became much more receptive to this, because it looked like a good design and it had a considerable rationale behind it. Now the novelty was the inexact exception that is raised when a numerical result has to be approximated, and you do an arithmetic operation and you have to approximate it. Maybe it overflows or underflows, so you have to approximate it. Or you have to round it and you have to approximate it. But if it's exact, don't raise the exception. And I'll explain what that was for in a moment. And the other was the NaN, the "not a number."

Now not a number existed in vestigial form on Seymour Cray's machines and also on the DEC VAX. The trouble was you never knew what it would do, and so you couldn't program with it. The purpose of the not-a-number thing was so that you could defer judgment. Something has

[41] The KCS document is described, in an interview of Kahan by Charles Severance as: "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."

happened that's bad, but maybe you won't use it because your program may branch. Whatever condition caused this to be bad may cause your program to branch later to a place where it just ignores this. That happens fairly often. It certainly happens with speculative execution. You know, you come to a place and you know you've got a branch but you don't know yet which way you're going to go, because it takes a while. So you run down both paths, but one of those paths may be fatal. You'll know that later when you'll be told, "Hey, you accept this path. Throw away that path. But in the meantime, that path, has hit an exception like divide by zero. Do you want to stop the computer?"

TH: So that sounds very much akin to what you were describing earlier in terms of throwing the flag but not immediately going to the operating system to hold things.

WK: That's right. You've got to give the programmer an opportunity to fix exceptional arithmetic events which are difficult to predict without doing extra testing that'll slow you down on the normal cases. So if on rare occasions something happens, you want to be able to test for it at your convenience. Then alright, so occasionally you're going to abandon this piece of computation or go over there and you do it in a slightly slower but more robust way. So we had inexact and NaN, but otherwise it was the divide by zero, which really means you create infinity out of finite operands. There's the overflow condition, which means you tried to compute something that was too big and now there's some interesting questions about what to do about it. And underflow, well, I made underflow gradual. Gradual underflow wasn't my invention. However, I had implemented it on the 7090 and the 7094, and it turned out to be really handy for some things.

TH: So underflow is the opposite of overflow, then the number gets too small and it vanishes.

WK: That's right, yeah. If you think of it in terms of digits, you imagine that your digits are pushed ever further to the right until finally they get to the end. The computer isn't infinite so, after all, digits have to fall off, and if you push it far enough to the right, they all fall off.

Well, gradual underflow was a way of arranging that the most common and frequent kinds of underflows could be handled relatively humanely without bothering people about it. You'd get an overflow signal, but you could ignore it, which is what we people did anyway because they're going to ignore an underflow signal no matter what you do. You can flush it to zero, you can do it gradually. People get in the habit of ignoring the underflow flag. So our task is not to correct a mistake but rather to treat underflow in such a way that the chances are pretty good that you could ignore it safely. That's the best we can do. And we'll raise a flag so that if you're a curious programmer and you're worried about it, yes, you can find out that it happened and then fix it up later.

So we had the exceptions, we had directed roundings. That meant that the normal mode of rounding was to round off within a half a unit in the last place. If it's half… Well, let's put it this way. If it's bigger than a half, you go up; smaller than a half, you go down; and if it's half, you go to the nearest even. Why to nearest even? Well, a statistician by the name of Hotelling had discovered that if you always round away from zero at the halfway case, you're going to introduce a weak statistical bias. And being a statistician, he said, "However, if you round to the nearest even" – or round to the nearest odd, choose one or the other, but round to the nearest even seems to work better in most cases – "well, then you don't get this statistical bias. Now the law of averages is on your side." So that's what I did. And it wasn't very difficult. We got guard, round, sticky. It really is easy to decide to do that, and the logic isn't particularly expensive. Of course there were people who said, "Oh my God, performance of this thing is going to be awful."

George Taylor was a graduate student and he built a board to replace the VAX floating point with IEEE standard-confirming float and double. How do you like that? And if anything it went faster. It was a marvellous design. They wrote it up for one of the IEEE symposia and it was going to be tested by Cary. What was his first name? Cary, C-A-R-Y. That was his last name. But he had a swimming accident. He dove into the pool and hit his head and nobody noticed that he was underwater for a while. So the guy who was going to do the tests didn't. So we really couldn't promulgate the board as a tested device even though initial tests had seemed okay. That was really… It was a very discouraging blow. There were a lot of us, not just his family, who were really grieving over that loss, because he was a nice kid. Ah, well.

Ultimately the tutorials, which is what these meetings had turned into, these tutorials bore fruit and we gained pretty much of a majority. Didn't get everything I wanted. I wanted to have two ways of treating infinity, but the engineers just choked at the idea that plus infinity might be regarded as equal to minus infinity. But it is, if you think of the real numbers as being on a circle and you have infinity up here. Both infinities fit. They're both the same. And you only need one zero. But if you have two infinities, you see, if you've got plus infinity and minus infinity, then you need two zeros, because what's the reciprocal of infinity? Well, it's zero with the same sign. That's a little bit troublesome. You've got two zeros and they're arithmetically equal, and the only way you can tell them apart is either by asking about their signs or dividing by them. But they go silently through computations and they allow physicists to deal with a common situation, which is you take a domain and you slit it, and when you slit it, the question is "To which side of the slit does the slit itself belong?" This way it could be both sides, and that gives you a closed domain you see, because it's closed on both sides.

So there were these interesting little things. How do you compare NaNs? And the answer is a NaN compares as unequal with everything including itself. And so on. So we had reasonable rules. Now you could predict how NaNs would propagate through your code, you could predict how they would affect your branches, and that was new. The indefinite on a CDC machine or on a DEC VAX, you could never predict what it was going to do. In fact on a DEC VAX, the chances were pretty good that if you touched it, it would trap, unless you just moved it. But if you tried to do an arithmetic operation on the reserved operand, you'd get a trap, and then what would you do? I mean you don't know where you are. That was part of the problem with the operating systems at the time. When you did your trap, you might have gotten a hexadecimal address, but the hexadecimal address couldn't be correlated with your source code because the compiler would rearrange operations in order to take advantage of pipelines and other things. Your instructions look like this, but when you look at them in assembly language, they're all jumbled up in order to be executed faster. I mean, isn't that what you want the compiler to do for you? So of course you couldn't figure out, if an exception occurred, where it had occurred unless people did something more than just turn off[42].

Well, you're getting some idea then of the IEEE standard. We settled on a final draft in 1981, but it wasn't adopted until 1985 because DEC opposed it. They brought a bevy of lawyers in to argue in front of the IEEE people[43] and so on. However, while they were arguing, it was being implemented. Weitek implemented it. I mean Intel of course came out with it in 1981 I think, and I think AMD shortly after. I forget how many… I don't remember the details of how many people there were, but there were at least six implementations. So DEC's lawyers' arguments got undercut. They said it couldn't be a standard because it wasn't in general use. But as it happens, by 1985, it was. And so it was canonized then.

It was a sad story for DEC because Mary Payne, who was their chief numerical analyst, was actually pretty competent, but she got bad advice from the hardware people who said that implementing this thing would be impossible. Especially gradual underflow, they said it was impossible to do that at a reasonable speed. They were mistaken. George Taylor did it.

TH: Did DEC ever adopt the standard?

WK: Yes. DEC produced the DEC Alpha. That was a pretty marvellous chip. Instead of waiting for things to settle down at the outputs of registers, they hand-tuned it so that the output of one register would go to the next without waiting to settle down, and that way they got rather higher speed for their technologies, for their line widths and so on. For their transistor densities, they were getting extra-high speed. And on the DEC Alpha, you could choose either to be compatible with DEC VAX floating-point arithmetic or with IEEE standard floating-point arithmetic and most of the other features of the IEEE standard. It was a little bit peculiar because the directions of rounding, whether you round up, down, or towards zero or so on, that was in the opcode and they didn't choose it well. They chose the… Oh, how shall I put it? They chose I think the zero, zero pattern for round to nearest, and what they should have done was have the three directed roundings and the fourth from the control register. If they had done that, then that would have been a really good system, a very satisfactory system. But they just didn't quite understand what to do, so they didn't quite get that right. But the rest wasn't bad. And I think the DEC Alpha continued in production on an Intel fab line for a while after the company DEC had disappeared.

TH: So we'll return to more recent developments with the standard a little later. We should say something about the Turing Award at this point. So you won in… Well, you won the award for 1989. I guess it would have been announced in 1990, the way they do it. The citation, "For his fundamental contributions to numerical analysis. One of the foremost experts on floating-point computations. Kahan has dedicated himself to 'making the world safe for numerical computations'!"

WK: Well, that was for a number of things I think, because they would have gotten word that a paper that Golub and I had written in 1964 made it possible to compute the singular value

[42] "….turn off the execution of the program" added after the interview.
[43] "…the IEEE people that the draft was just an unimplemented proposal" added after the interview

decomposition of matrices of not immoderate size much faster and more accurately than previously. And I had written a number of other papers on things like "Why does something converge the way it does?" or "How do you compute this without losing digits?" A transcendental function library was produced by some of my students for the freely distributed math library now. That was in the early 1980s. That included argument reduction for the trig functions sin, cos, and tan as if we knew π to infinite precision, but we didn't. However, a student, Bob Corbett, found a way to do it which required only about twice the precision of what we were using. You know, working precision is this wide and if you carry that many digits, you can reduce your arguments as if π were known to infinite precision.

And a similar scheme was discovered by Mary Payne and her workers at DEC, except they needed twice as many bits as we needed. There was a guy, ah, Vyssotsky, Vyssotsky[44] at Bell Labs who didn't believe that we could do it. But he tested it and he found yeah, it worked. And it worked because I'd found some continued fraction manipulations, and another student, Stuart Macdonald, he helped to write a program that first got a continued fraction due to… Was it Robert Gosper from MIT? He got his continued fraction for π and we used it to show that we didn't need more than this many digits. So it was fast, it was economical.

So we had a really nice library and guys at Sun continued to develop it. The guys at Sun, well, many of them were my students, former students, and they continued to develop it until Oracle took over. So I think you can get the last version that Sun distributed, but there was an even later version. I think Oracle has not distributed that. Oh well. You can be grateful for what you can get, I guess.

TH: And so this is some of your other contributions, and in the long interview that we did back in 2005, you gave a much broader perspective on those things than will be possible here. Were you surprised to win the award?

WK: Well, I certainly hadn't expected it.

TH: That sounds like surprised.

WK: Hmm?

TH: That sounds like surprised then.

WK: Well, yeah, I guess so. I didn't do it for the award. I did it because it seemed like the only good way to go. And I had a lot of help. That was one of the things that the award didn't make as clear as I wish it had. It wasn't just Jerome Coonen, a graduate student who worked with me on the standard. I had a number of friends and colleagues… Well, you've spoken with Jim Cody. He certainly helped. Then there was Fred Ris at IBM. IBM didn't meddle in the standard. They

[44] Victor Alexander Vyssotsky (1931 – 2012) was the technical head of the Multics project at Bell Labs and later Executive Director of Research in the Information Systems Division of AT&T Bell Labs.

behaved as if they were above it all. But Fred Ris was nonetheless an important ally. And I had some reluctant allies from Hewlett-Packard. They didn't get quite all of what they wanted, but otherwise they were happy to see it going that way.

Then of course there was George Taylor, the graduate student who actually built the thing. You see, he build a high performance board to replace the floating point board on a DEC VAX, and what's more, he did it twice because there was a machine called the Elxsi – E-L-X-S-I – I think it was 6400, and he built IEEE standard-conforming floating point for that machine. It was a supermini that was actually faster than a DEC VAX.

TH: So the citation itself doesn't mention the floating-point standard, but it sounds as if your sense is that that would the primary thing the award was given for?

WK: I think so. Yeah. I mean normally mathematicians don't get awards from the ACM, unless they're working in computability theory or something like that. But really the work on floating point had an enormous commercial impact. It meant first of all that people could exchange data between even hostile machines. Even big endian versus little endian. As long as you knew which was which, you could exchange the bit patterns and you'd get exactly the same numbers on this machine as you had on that machine, until of course you started doing the arithmetic because then there were variations in how people optimized things and what instruction sets they used and so on. And it also meant that if you stuck to a certain core set of operations, you could pretty much guarantee that the results would be exactly reproducible. Until we got threads, until stuff started happening in parallel, because when it happens in parallel, you can have a whole bunch of different threads and they come together each one in its own sweet time, and when you add them all together, the order in which you've added the numbers depends upon which thread got there sooner. Well, so now you lose reproducibility and you have to use special codes which are being developed, have been developed by students in my college. Jim Demmel, a former student, and some of his students have developed algorithms for dealing with that problem. But before parallelism became popular, if you stuck with the same core instruction set and you used the same math library – which you can get, FDLIBM[45] – you can get exactly reproducible results. They might not be correct, but at least you're in the same boat.

TH: So if you could write your own two- or three-sentence citation, what would it say?

WK: Citation for what?

TH: Well, citation for the Turing Award.

WK: Oh, for the Turing Award. I see. It would say that I was very lucky in my choice of collaborators and opportunities to be able to get something done that was really worthwhile, but I didn't do it myself. And it's unfortunate that when you get these awards, it makes it appear as if "This guy's a hero. He did it all himself." Well, I've got a bowl and I think I got $25,000. Whoopee! But I was in some respects just a – how shall I put it? – I was a guy near the centre of it all for a while, but I wasn't the only guy working on it. Jerome Coonen had a much better

[45] FDLIBM (Freely Distributable LIBM) is a C math library. for machines that support IEEE 754 floating-point arithmetic.

political sense than I had, and so he often tamped me down a little bit. You know, moderate things, moderate things. "Don't irritate people unnecessarily." And Jim Cody had an enormous amount of experience because then Argonne had so many different machines.

And then there was Stan Brown at Bell Labs. Stan Brown came up with a model of floating-point arithmetic which ideally would have had the property that if your program uses only those axioms that Stan Brown articulates, then you should be able to prove your program works correctly if it can be proved at all. You hope. But that turned out not to work out, first because his axioms didn't cover the CDC 6000 family. I don't think it covered all of Cray's either. But here was what was… in order to cover all the different computers, the axioms couldn't be categorical. They limited the possibilities, but they didn't specify the possibilities. That meant that it's like having the design of a computer whose arithmetic is worse than any arithmetic ever actually built. You've got all these inequalities. What are you going to do with them? He came to appreciate this limitation of his scheme. It had looked so good at first, as if now at last we have an axiomatic foundation for floating point. And all it did was give you an opportunity to be unable to prove two theorems about the actual arithmetic. So that was a valuable lesson, an important lesson, which meant that a standard could not be descriptive, it had to be prescriptive.

I could go on and on with various bits of experience culled from this place or that place, people who contributed to these things in many ways, and I just happen to have been around to have sort of accreted all this stuff.

TH: And do you feel that winning the award gave you any opportunities to have your voice heard or to do things that you might not otherwise have been able to do?

WK: Well, you might have thought so, but as I've said, the importance of floating-point computation has dwindled. When I started computing, the *raison d'être* for a computer was to do scientific and engineering computations. Specifically they were worrying about dredging the Saint Lawrence Seaway and where are you going to put the mud? That was the sort of computation they were doing. That was a civil engineering computation. That's what the machines were for initially, especially since they were binary, so the business world didn't think that that could be of any use. Well, it was, but they didn't think so. And over the years, although scientific computing has grown enormously, spread enormously, relative to the rest of computing it's dwindled to a little raindrop.

So you've heard me say it, numerical computing is like a sliver under the fingernail of computer science. It's just an irritant because there are all these things that you have to watch out for that don't occur if you limit yourself to computation with integers or with character strings or with discrete structures like graphs, until you start approximating graphs, because they get too big. And when your graph has a billion nodes, ah, then there you are approximating again.

TH: So we should say something about what you've been doing since the award. I know a chunk of that is the revision of the standard, but there are maybe some other ….

WK: Oh yeah, that's a very painful thing. The standard is supposed to have been revised every five years, but in fact the revision didn't start until 2000. Fifteen years. I think people were afraid to tamper with it. But that revision came under almost a direct order that we combine the binary floating-point standard with a decimal floating-point standard. And there was a guy in England who worked for IBM. Oh God, his name has just dropped into a hole in my memory.

TH: Now would combining the standards be so that there's only one standard to maintain? Or would the implication be that chips should be able to do both to meet the standard?

WK: No. What it amounted to… Oh, it was Mike Cowlishaw – C-O-W-L-I-S-H-A-W. His design for decimal arithmetic promulgated through a language that he was fond of at IBM in England, it allowed not only for floating-point computation in decimal very much as it had been done on the IBM 650, although better rounding properties. He also wanted to be able to use this decimal arithmetic in contexts where what you were really dealing with is essentially integers. So, for example, in most cases billing dollars and cents or pounds and pennies are really in integer multiples of pennies. What he wanted was to arrange that people could do that type of computation very fast, which meant without normalizing the floating-point numbers. So the floating-point number might look a little bit anomalous. You've got a string of digits and then a large bunch of zeroes and your exponent. And he had worked out a scheme for supporting that. But his scheme would have meant that decimal arithmetic in a given word size would have either a rather smaller range than binary or a rather smaller precision, or both.

But Dan Zuras – Z-U-R-A-S – he had retired from Hewlett-Packard. He had been a hardware engineer for them. He and I came up with a packing scheme that meant that although it wasn't all that easy to read, the bit string that represented a decimal number, because chunks of it were compressed, you now had in a given word size a range and precision comparable to the binary format of the same word size. So you wouldn't sacrifice range or precision significantly, although there is a well-known wobble that's associated with decimal arithmetic, and that is that it just makes it harder to do an error analysis, that's all.

Well, that would have been tolerable. But then there's sort of a bandwagon effect. All sorts of creatures jump onto the bandwagon. These were creatures who came from companies that had their own axe to grind. They wanted their own product to be standard-conforming and they didn't want to have to modify it. That induced a number of what I would call centrifugal proposals, proposals that detracted, pulled away from the centre. People wanted certain options to be allowed so that you'd have your choice about whether you'll do it this way or that way, because this company does it this way and that company does it that way, or wanted to do it this way or that way.

Well, the trouble with a standard is when you start putting in too many options, it's not a standard anymore. The outcome is unpredictable. And that's what has happened. This has caused real pain. For instance, Sylvie Boldo – S-Y-L-V-I-E, and then the last name is Boldo, B-O-L-D-O. Now she works for one of the French research outfits in one of the southern suburbs of Paris. And her accomplishment is to be able to use the theorem verification language Coq – C-O-Q – to verify the correctness of certain floating-point computations. But she has observed ruefully that although she's been able to do it for things that conform strictly to the IEEE standard, when she gets to programs where somebody exercises certain options, it invalidates her proof. It doesn't mean the code won't work – maybe it will, maybe it won't – but the proof doesn't work, that's for sure.

So that's, you see, one of the expenses that you incur when you have too many options. And the 2008 standard, what evolved after 2005, that has too many options. It has become very complicated. And part of the complication arises because nobody is willing to bite the bullet and say, "Who should conform to this standard? What should conform to this standard? Should everybody conform to it?" Well, if everybody's going to conform to it, then you have to have all these options, you see, and then you don't know what conformity means anymore. And I just couldn't get this across. Well, that may have been a failing on my part. I just failed to get it across.

So I'm not happy with the way the standard has turned out. I think the older standard, the 1985 standard is much more humane, even though it does have certain gaps. And here we are revising it again. We're on schedule to produce a revision at the end of this year. But the chairman of the revision committee, David Hough, a former student of mine, has said that he does not propose to incorporate anything new, only clarifications of the old. God knows it needs clarifications. And so it'll still be too complicated. And the decimal thing is so screwily written that I can't blame people if they don't understand what's going on.

I really should do my own write-up to explain what's happening, but I've got… A year ago, I fell ill because a doctor had told me that I have benign prostatic hyperplasia. What does the word "benign" mean? "Doesn't cause harm." No, that's not what it means. It means it's not cancerous. It wasn't benign. And I suffer from shrunken cartilages between my vertebrae. They pinched nerves, so half the nerves below my waist stopped reporting, they stopped functioning properly. That's why you see me hobble. I didn't know that the prostate was causing enormous pressure to build up and it damaged my kidneys, which is now why I have to have this little pipe, so that it'll drain my bladder and not allow back pressure on kidneys.

But all of that has taken a year out of my life, so I didn't get around to writing up what I should have liked to write up to explain the decimal part of that standard in intelligible terms. Because it's really two arithmetics in one. It's an ordinary floating-point arithmetic, just as you'd imagine it to be with all the digits shifted as far as possible to the left, to normalize, and then you have an exponent (so in scientific notation, as scientists say), combined with a fixed-point format where now your decimal point or quantum point is on the right-hand side and you're dealing with numbers that are all jammed to the right as far as they can go, because that's the natural way to write them. And there is a transition point where if the numbers get too big, you end up with floating point and rounding occurs. And I hope you look at the inexact flag because it will warn you that you may have lost some pennies. Or who knows? Nowadays you may have lost some hundreds of dollars or millions of yen.

That part is not what bothers me. What bothers me is all the other things that people wanted to stick in. Kulisch, Ulrich Kulisch at Karlsruhe in Germany has people who like his ideas and they wanted to put in what's called a superaccumulator, which is a register so wide that if you multiply two numbers and then convert them to fixed-point, they'll stick somewhere in this wide register. It allows you to do some things which can also be done in other ways, but this happens to be the way that Kulisch and his students do it.

So there's an option, and I can go on and on with various options.

TH: So are those features that have actually been adopted by any microprocessor designers or hardware…?

WK: Well, except for research purposes, that one hasn't been. There's a fused multiply-add. That's partly my fault. In 1984, I went down to Austin, Texas to help with a problem that had been incurred because IBM wanted to implement John Cocke's very reduced instruction set computer. You know, R-I-S-C, RISC, a very, very popular buzzword. Actually the first RISC computer was Seymour Cray's 6000 family, but that's a story for another day. The trouble was that John Cocke had observed that all they really need is a multiply-add operation, an operation which multiplies two numbers, rounds them, then adds a third and rounds them. He said, "If you've got that, you've got everything you need, because if you want to do a multiply, add zero, if you want to multiply… sorry, if you want to do an add, multiply one of the numbers by 1, and if you want to do a divide, well, you'll use an iterative process." Oh, it wasn't quite that simple because if what you want to do is round correctly the way the IEEE standard specifies, rounding to nearest, ahhh, then it will cost you twice as much as just simply getting a quotient that's OK, except maybe for the last bit or two.

So that's where they called me and said, "What can we do about this? We can't afford divide to be that slow." And I looked at their chip and I said, "You know, you've got room here to do a fused multiply-add it says. Do the multiply, don't round it. Then do the add, and then round it." And I said, "If you do that, then you can compute a remainder, and from the remainder, you can figure out how to round it, and you won't have to do something that's so lengthy." And then I said, "And besides, it helps with decimal-binary conversion and it will help with your logarithms and other things too." And they did; and it did.

And Peter Markstein, he had worked for IBM in Yorktown Heights. That's where I first met him, but he was working for this group in Austin. And yes he, and I think with the help of his wife, they produced a math library and they found that the fused multiply-add made all sorts of things go better. It was almost like having a scratchpad, you see.

And well, Robert Montoye. Robert Montoye was the engineer who actually implemented the fused multiply-add. I found out later, years later that his thesis had been about fused multiply-add, and it's just possible that he left room on the chip hoping that a visiting fireman like me would say, "Hey, fill it up and we'll use it for a fused multiply-add." Maybe. But I've asked about it and nobody seems to know. That is a crock for the historian, because I bet you would really like to know, and I would like to know, whether he was serious about this.

Anyway, it got done. And then the question was "Well, does it conform to the standard?" I said, "Well, it goes beyond the standard. And as long as you let people know that you're using it and let them know that they can turn it off, that's okay, because," I said, "there are some situations where you'll wish you hadn't done this." An example is multiply a complex number by its conjugate to get the squared magnitude. You'll wish you hadn't done this, because when you do it, you're going to find the squared magnitude, which should be real, has a tiny imaginary part due to roundoff. Bad news. And there's some other peculiar situations where this happens.

So I said, "As long as you can turn it off, well, it's something that somebody can use if they want, or not." Then other folks decided that they'd like to have it too, and so there we are. It's another option in the standard, in the 2008 standard.

TH: Has the 2008 standard had any effect on what vendors are doing with real machines at this point?

WK: I don't think people have departed from the 1985 standard, to my knowledge, unless what they've done is depart from the standard altogether. For example, ARM, A-R-M, they have processors which do floating point in a way that does not conform to the standard. It doesn't round quite right. It doesn't have flags. The exceptions are very stilted. There's certain rudimentary options. I think they even underflow abruptly instead of gradually. Oh, abrupt underflow is now an option. Some people want it. I don't think it's a good idea, but if they want it, well, what can I say?

No, I think that the old 1985 standard is the one that's still preponderant in hardware. But that's fading because the languages are slow to support it. There's a version of C, C99, that supported the IEEE standard to some reasonable extent, and the latest revision for C is underway to support the standard more fully. There's also a version of FORTRAN which, if I remember rightly, makes provision for access to flags, but it thinks that they're machine bits. It doesn't understand them as localizable variables. And there are serious questions about when you enter a subprogram, what should you do? Should you inherit the flags? Should you propagate them? If you raise flags and deal with them in your own subprogram, what should you put out?

APL dealt with this problem. I don't know if you remember the APL language, Ken Iverson's language. Well, what they had was a situation where if you entered a function, you would inherit what were called system variables. System variables include a comparison tolerance, CT, and included an index origin, could be 0 or 1. So if you entered a subprogram, you'd inherit. Now if you declared these as local variables and altered them, then when you exited, you'd restore things the way they were when you entered. But if you didn't declare them as local variables and you exited, well, then whatever you may have changed, that's what you passed on. That's not a bad way to handle flags and the directed rounding modes. But that's not been implemented so far as I know in any current languages, and APL is a ghost. It's long gone. Well, I have an APL implementation for one of my computers, but I hardly ever use it.

TH: Yes. It had a devoted following, but it was never a very mainstream…

WK: Say again.

TH: APL had a devoted following, but…

WK: Oh yeah. There were people who loved to write programs in APL. The trouble was you couldn't read them.

TH: So are there any other aspects of your work over the last few decades that you think it would be relevant to mention?

WK: I'm having trouble hearing you.

TH: Sorry.

WK: Are there any other aspects of my work…?

TH: Over the last 20 years or so that you think it would be relevant to mention?

WK: Well, I've worked on all sorts of things, as I always do. But what weighs most upon my mind is our inability to debug our large floating-point codes, and I've proposed what I think should be done in a posting called "Boulder.pdf" – that's the name of the city in Colorado. And this includes what amounts to two, let's say two major foci. One is "How do we deal with rounding errors?" We can't name the rounding errors in our programs because there's so many of them we'd drown if we tried, so they're invisible. And when people write programs and they write out algebraic expressions and they imagine that that's what the computer is going to evaluate, that's not what's going to happen. Every operation is going to get rounded. Well, some operations like division by 2 or multiplication by 2 hardly ever are inexact, they hardly ever over/underflow, but they're exact as far as roundoff is concerned. And there are lots of such things that are exact, but usually there's going to be a rounding error. The rounding error of course affects only the end figures of the immediate result, but then that can turn out to have a really large effect. Depends on where the singularities of your program are. And the singularities are very hard to find. I've given examples where to most people's eyes, the singularities are invisible. Oh, they're there. They kill the accuracy of the result. But when you look at the program, you can't see why that should happen.

So let's face it. Rounding errors are going to baffle even experts from time to time, and therefore we need tools to deal with rounding errors. And after a lot of experiments and after trying a lot of other suggestions, other people's suggestions and so on, I've come to the conclusion that the least expensive way that offers the most for your patience and time is to rerun a computation three more times, one for each of the directed rounding modes. That gives you four results – round to nearest, round up, round down, round to zero. And very often – I can't say "almost always," though I'm tempted – very often the spread in those results gives you an idea of the sensitivity of your computation to roundoff in the internal computation, the internal works.

Now it can't be infallible, because I also give examples showing where it won't do you any good. But the examples are pathological. They're designed specifically to show you that this scheme is not foolproof. But the computations don't compute anything you'd want to compute. They're just there to show that you can't always guarantee it.

Now that sounds easy, but it's not, because for instance, do you want to redirect the roundings inside the math library? Probably not. And in order to rig these tests, what you want to do is identify some module in your program, like one of the blocks in the block diagram, you want to identify some module that seems to be hypersensitive. In order to do that, you got to capture the input that came to this program at a time when your suspicions were raised, and then rerun it and look at the spread in the output. And if that spread is excessive, you can say, “Ah-ha, I think maybe this block is responsible for my troubles.”

But you can’t do that without assistance from the debugger. That means the debugger and the compiler have to collaborate, so you can tell the debugger, “I want to save the input to this subprogram in order to be able to run these tests. And I want to protect this logarithm and exponential program and decimal-binary conversion, and so I want to protect them from redirected roundings. I always want to protect decimal-binary conversion from redirected roundings for this purpose.” And so the debugger has to be able to insert the appropriate things into the running code, the object code.

Now there’s another way to do it, and that is you’ve got a program let’s say in float arithmetic and maybe what you’d like to do is run the same program text-wise except that all the declarations are now in double. In C, that’ll tell you what’s gone wrong. Now if what you want to do is trace the programs in C where they diverge, you’ve had it because that’s not going to tell you anything. It turns out there are lots of programs where the results diverge as a consequence of rounding errors, but they always come back. So see, you’d get a false positive then.

No, I show how to do it in that document. I show how to do it so you can take advantage of the ability to run with extra precision. But that’s not trivial either because some of the subprograms you’ve obtained have been obtained from other people in object form and you can’t recompile them for the higher precision, and some of the constants, some of the parameters have to be modified if we’re going to go to higher precision. I mean if this is an iterative program, do you want to iterate until convergence to a higher precision or do you want to iterate just the same number of times as you iterated here? Well, it depends, so you’ve got to be able to exercise these options in order to be able to benefit from running the higher precision. Neither then of these schemes is altogether trivial. They require support – the language, compiler, debugger, and in some cases the operating system.

So that’s one arena, dealing with roundoff, the invisible menace. Now the other arena has to do with exception handling. Floating-point exceptions are different from errors unless you handle them badly. The reason they’re different is because there’s so little that you can do with them.

Okay. What can you do about a floating-point exception? Well, you might decide to abandon the computation. That’s the try, catch, and so on. But a graduate student here, now he’s a professor I think in Virginia, Weimer – W-E-I-M-E-R, that’s his last name – he found out that if you use try-catch-finally for more than one condition to be caught, the chances are pretty high that you’ll do it erroneously. He looked at some vast number of lines of code, maybe a million. I don’t know how he did it, but apparently he set up a search procedure. What he found was that more often than not, if there was more than one condition to worry about, the programmer got it wrong. That would happen with floating point also, because with floating point, it’s very rare that you have just one anomalous situation.

So as a programmer, you should certainly have the option to try something and if you detect that it doesn't work, then go back and try something else. You should have the ability to presubstitute, which means you should be able to say at the beginning of a block, "In this block, if such-and-such an event occurs, substitute this number." All you have to know are the operand values and the destination and the operation. Then you look up a table and you should be able to set whatever you like in that table. It can be either a value or the address to a value. And the operating system should not get into the act unless you've decided what you want to do is really abort. Maybe if what you're doing is debugging a program, that's what you may indeed want to do. Then you hope that if it aborts, it will tell you at what line so you'll know where the exception occurred, not with a hexadecimal address but with a line in your source code. And that means that that information has to be preserved by the compiler and available to the debugger. And possibly the operating system has to get into the act, because if you're going to abort, where are you going to abort to? Well, you can abort to a calling program or you abort to the operating system. So operating system has to be prepared.

Now there is a larger problem with exception and error handling that has nothing to do specifically with floating point. It has to do with the whole concept of a runtime error. Because it's ambiguous. Is a runtime error an error on the part of the program's user or is it an error on the part of the programmer? Hard to tell. What that means, as I see it, is that especially in real-time systems that are running something like an aircraft, you've got to think your way through very carefully so that before a program abandons control, it first of all finds a default situation which is as little harmful as you can imagine it to be in a world where you can't imagine everything, and you provide a message which is usable at a higher level to say, "What went wrong?" And if what went wrong was potentially transient, programs should maintain a watch to see whether the transient has passed and now things are okay.

I say this because if you don't do that, it will cost lives and it has. If you look at my "Boulder.pdf", you'll see it happened to Air France 447. In 2009 I think it was, it got lost in the middle of the Atlantic on the way from Rio de Janeiro to Paris. It got lost because it was caught in a thunderstorm, the pitot tubes iced up, they stopped sending reliable airspeed indications. The automatic pilot said, "How can I be maintaining my altitude at 35,000 feet when I'm only doing 160 knots or less?" So the automatic pilot said, "Invalid data. You guys take over." Invalid data? What data was invalid? The pilots didn't know. Which of their instruments must be distrusted? The pilots didn't know. Could they look outside and see what's going on? No, it's pitch-black night and they're being tossed up and down, and they can't see anything outside the window.

Well, the younger co-pilot figured that if he doesn't know which way we're going, up is better than down, so he pulled back on the stick. That turned out to be a mistake, because when they pulled back on the stick, it increased the angle of attack of the wing and the aircraft stalled, and it started to fall like a rock. Then there was a signal that said, "Stall, stall, stall," but can you trust it? You look outside, you can't see what's going on. You're being tossed around anyway in a thunderstorm, so you can't tell by the seat of your pants. It took them more than three minutes to figure out what was going on. Three of them, two co-pilots and the captain. It took them more than three minutes, but it took them only three minutes to hit the surface of the ocean.

So they and all the passengers died, and they died because they didn't have an informative indication of what was wrong. Because as they fell down through 20,000 feet, the pitot tubes lost their ice and started to record correctly, but the program had abandoned them. It was no longer looking. This is a tragedy that was caused by software, but the board of inquiry didn't ask about software. They ended up blaming the younger co-pilot for pulling back on the stick when what he should have done to get out of a stall, if he had known that he was in stall – and certainly the plane was trying to tell him he was in a stall, he just didn't know he could trust it – you push forward on the stick. You put the nose down, you gain speed. When you gain speed, the stall condition goes away, and now you can pull out and continue in a normal way. Of course you can if you know what to do.

Well, they lost another aircraft on the way from I think somewhere in Indonesia to somewhere in Malaysia. This was not the one that was supposed to go to Beijing and ended up in the Indian Ocean. No, this is the one that was… it was a two-hour flight, and in the middle of the two-hour flight, they got caught in a thunderstorm. Inexplicably, one of the pilots popped the circuit breaker that supplied power to the automatic pilot, which I interpret as meaning that that was the only way he knew to reboot. But it takes time to reboot, and during that time, they didn't know how to control the aircraft. So it crashed. They lost it. They managed to recover I think the black box, but the pilots and all the passengers got lost. Same thing. The software did not tell them why they were being abandoned.

I think we have to look at programming languages and ask ourselves whether a programming language convention is worth enforcing. Namely that every programmer must have in mind that if something happens that he has not anticipated and it causes an error condition that causes a program to fail, that there should be a default status of the data structures that the program's responsible for, and that that's the condition in which the program will put them before it abandons. And in the calling program, knowing that, knowing that if the program fails the data structures will be consistent, they'll be in some neutral or standard or predictable format, it can try to deal with the emergency. But you can't just abandon the guys and leave the data structures, which means the control of the aircraft, in some unpredictable state and they don't know what it is, don't know what to do about it. That will cost lives.

Well, I haven't yet updated that document with the Malaysian Airlines thing, but you can read about that in "Boulder.pdf". That is the most important thing that I'm doing. It's not my favourite thing. In a way, I don't want to get involved with that stuff, but what can you do? It seems to me to be necessary.

Other things I'm doing are what an error analyst would do. There are people who want to compute the energy levels of various molecules, which are represented by differential operators, by differential equations, Schrödinger's equation. The differential operator has eigenvalues and they'd like to know the tiny eigenvalues. But the differential operator is an unbounded operator. That means that the derivative of a wiggly function can be an awful lot bigger than the amplitude of the function. Since the function would be arbitrarily wiggly, that means that the differential operator is essentially unbounded. So when you discretize it, of course discretizing it, now you've only got something finite. But this finite thing can get rather big, really big, because it's an unbounded operator, and you want the small eigenvalues. What's more, you'd like them to reasonably high accuracy.

This is a challenge. And, well, my collaborators like Jim Demmel and others have found ways to do the computation which usually gives you very high relative accuracy. You usually get practically all the digits you want from the small eigenvalues, even though they're small – compared with bigger ones, they can be enormously bigger. But how do you know? How do you check it out?

So I've been trying to compute error bounds which say, "Under such-and-such circumstances, you can confirm the accuracy of what you've computed so you know 'Yes, this many digits are okay.'" But it isn't often that you can do that because it isn't often you can get high relatively accuracy of the tiny eigenvalues. Usually the bigger eigenvalues determine the rounding error level. So you've got your big eigenvalues here and you look at their rightmost digits and you imagine them being somewhat fuzzy, and as the small eigenvalue gets smaller, after a while they disappear in the fuzz. That's what usually happens, so it's only in rather special occasions that you can get high relative accuracy where you get digits that go like that.

Well, and then I'm reviewing a crazy book that says… "THE END OF ERROR: Unum Computing"[46]. "The end of error" means that if you do what he says, you'll never get incorrect results. This is provably impossible because, as Derrick Lehmer said, once we acquiesce to approximation, to rounding errors, you're in a state of sin. And I can go into details, but I don't think that's what you have in mind right now.

TH: No. Alright. Well, you had mentioned that you would like to say something about the role of luck in your life.

WK: Yes. Well, I think there are lots of people who are at least as smart as I am and maybe a lot smarter, and maybe more virtuous and maybe more industrious, and they appear to get less recognition. The difference I used to think was that some guys are really a lot smarter than others, but I know better now. Luck plays a large role, a very large role. You've got to have the right opportunities. It's helpful to have the right friends. There are times when something is right and other times when it isn't.

And my luckiest stroke, luckiest by far, was persuading a young woman to grow old with me back in 1953, and that's what we've been doing. Without her, I'd be a lazy bum. In order to deserve her love, I had to change my ways from being relatively lazy, do-what-comes-easy, to being industrious and trying to do what's right. She has supported me, she has taken care of me. She soothes and stimulates me, and has for some 63 years. So God, I was lucky. Because she could easily have been snapped up by someone else. I look around and I see a lot of people whose marriages seem temporary. I guess they just aren't as lucky as I was, because I haven't noticed that they're stupid, I haven't noticed that they have defects of character that are obvious to me.

[46] CRC Press, 2015

So I think we have to acknowledge that luck plays a large role. I was lucky that Jerome Coonen wandered in on me one day and stayed to help out. I was very lucky that I was living down here instead of in Toronto so it was actually possible, practicable to participate both in the consulting with HP and Intel and IBM and a few others, whereas from Toronto it would have been harder.

I left Toronto partly because of the damage done by John Diefenbaker[47]. He was elected in 1957 on an anti-American platform. It's almost unbelievable that a Canadian prime minister could be elected on an anti-American platform under the circumstances, but that's what happened. How did it happen? Well, he came from the Prairie provinces and they grow wheat. And it's a very good wheat. Durum wheat. It's very hard, makes very good flour. Now of course processed flour may not be all that good for you, but back in 1957 people thought it was just marvellous. And the United States subsidized American wheat farmers' exports, so of course Canadian farmers got testy about that, and Diefenbaker was their champion.

As far as he was concerned, the farmers, the miners, the lumbermen, they were the salt of the earth. Those businesspeople and scientists and especially the Liberals… He was a member of the Progressive Conservative Party about which nothing is progressive and they didn't conserve anything except their family's wealth. But anyway, he was a member of that party and the Liberals then of course were scoundrels as far as he was concerned. And as he campaigned, the Senate of the United States was debating laws that would restrict American defense purchases to American producers, but there was an exemption for Canada as there had been during the Second World War and continued into the '50s. But when they heard Diefenbaker spouting off, they got into a snit and they struck out this exemption. Now of course the exemption could be put back. I mean Congress can change its mind if it feels like it.

That had an impact on the Avro Arrow. The Avro Arrow was a jet interceptor designed by people living in Canada though working for an English company. You see, we had different immigration policies. Canada allowed immigrants to come if they brought with them various technical skills that the government deemed to be in short supply whereas in the United States there was a quota system. The H-1B system hadn't really been perfected yet, and so there was a quota system. Well, you could be in a country with an unlucky quota and you got several years to wait before you'll be admitted or even considered for admission. So many of the European technologists, especially the Germans and British came to Canada, and we were happy to have them. That included people who were experts in aeronautical engineering, in ceramics, materials for jet engines, and on and on.

And we needed this aircraft because people were worried that bombers from Russia would come sneaking over the Arctic in order to bomb American cities. Well, of course the Americans would have to intercept them, but the range of American fighters was such that these bombers would be intercepted over Canadian cities. We didn't feel happy about that, so we were willing to pay taxes to have an aircraft designed that had the range to intercept the bombers way up near the

[47] John George Diefenbaker (1895 – 1979) was the 13th Prime Minister of Canada, serving from 1957 to 1963. Always known as "Dief the Chief," he was not shy about cancelling projects started by his predecessors.

Arctic Circle where hardly anybody lived. This aircraft was capable of supersonic speeds, capable of very considerable range. It had two very powerful engines. These were Canadian-designed "Orenda" jet engines which were certainly the most powerful in the Western world if not in the whole world. And three Arrows had been built and two of them had flown.

But Diefenbaker got elected. He was not quite the Donald Trump of Canadian politics, but it was similar to that sort of phenomenon. The first thing he did was have the Avro Arrows cut up into scrap, just like that, because they represented to him the Liberal Party. He foamed at the mouth when he talked about the Liberals.

Well, what that meant was that all these people in the small machine shops and the ceramic shops and the aeronautical design shops and the computing and everything else, a lot of it around the bypass highway near Toronto, all of them realized that they weren't going to be employed anymore. And Americans realized that too, so Northrup came up and they bought the airframe. And Pratt & Whitney came up and they bought the engine. And Hughes came up and bought the electronics and so on. But they didn't just buy the parts, they bought the people.

What has to be understood, and it isn't understood enough, is that you may think you've got high tech in patents on your shelf, but the high tech resides in the minds of people. It doesn't do any good on paper in patents, or for that matter in PDF files. And if you don't provide a path for your young people to progress, then they're going to go somewhere else, and they'll take your high technology with them, non-compete agreements notwithstanding. So of course that's how we got the British, and the Germans came because things were tough in Germany. We also got others – Italians, French, and so on. So we had all these high-tech people and in a fortnight, in two weeks, they were gone, snapped up by American companies and transported.

And I didn't realize at first that this was going to impact my students, because I was still in Cambridge when it happened. But when I came back in 1960, I could see what had happened, that the jobs available for my students were available preponderantly south of the border. I felt somewhat discouraged about that. Maybe I should have stayed and fought it out, but it just seemed like… it seemed so overwhelmingly difficult to fight it in that particular political and economic climate.

So when I got an invitation to come down to the United States, I thought, "Oh boy. I'll come down here for seven or eight years. By that time, maybe Canada will have recovered." Well, it didn't. And after seven or eight years, when I asked my family, "Hey, we've been here eight years. How would you like to go somewhere else? We could go to London or to Adelaide or to Boston. There are all sorts of places where we could go. How would you like to get a fresh change of scene?" "No, no, no!" Three votes no. My two young boys were now old enough to vote, and my wife was happy here, had made lots of friends. So I was stuck.

Now I didn't suffer. The decision to stay here has not created any significant disadvantage for me. Quite the contrary. But I feel sad that Canada lost another brain. Don't have that many, you know? Well, they've got a bright young Prime Minister now, young Trudeau. He seems like a pretty nifty guy and maybe he can reverse or at least start the reversal of decades of this government. I think a country that does not provide a role for the young people to gain experience, to make mistakes, is wasting them. It is part of it due to not doing your duty. And you asked me what I thought was important, and I've told you another one of the things I think is important.

TH: So that I think concludes the outline that we had. Is there anything else that you would like to mention?

WK: Well, I don't know. I think I've kept you for long enough. It's getting very foggy out. I hope you'll be able to find your way home.

[end of recording]