An interview with

WILLIAM M. KAHAN

Conducted by Thomas Haigh

On

August 5-8, 2005

In Berkeley, CA

Interview conducted by the Society for Industrial and Applied Mathematics, as part of grant # DE-FG02-01ER25547 awarded by the US Department of Energy.

[Version 1.1, revised March 2016 with corrections from Prof. Kahan]

Society for Industrial and Applied Mathematics
3600 University City Science Center
Philadelphia, PA 19104-2688

ABSTRACT

William Kahan discusses the whole of his career to date, with particular reference to his involvement with numerical software and hardware design. Kahan was born in Canada in 1933, growing up around Toronto. He developed an early interest in science and mathematics, tinkering with mechanical and electronic devices. Kahan earned a B.A. in mathematics from the University of Toronto in 1954. He discusses in detail his experiences with the FERUT computer from 1953 onward, including its operation and use and the roles of Kelly Gotlieb, Beatrice Worsley, Cecily Popplewell, Joe Kates, and others. During the summer of 1954 he worked with Kates to produce a prototype airline reservation system for Trans-Canada Airlines. Kahan then began work on a Ph.D. from Toronto, graduating in 1958 under the direction of Byron A. Griffith. He explains the state of its mathematics program and curriculum at this time, and outlines his own thesis work on successive overrelaxation methods and the beginning of his interest in backward error analysis. Kahan spent the summer of 1957 at the University of Illinois, where used the ILLIAC and met Dave Muller, Don Gillies and Gene Golub. After graduation he spent two years in Cambridge, and he recounts his experiences using EDSAC II and interacting with Maurice Wilkes, Jim Wilkinson, Chris Strachey, Alan Curtis, and J.C.P. Miller. From 1960 to 1968 Kahan served as a faculty member at Toronto. He discusses in some detail his work with the IBM 7094 installed there, including efforts to rework its compiler, operating system, and elementary transcendental function library to improve the support of numerical calculation. Kahan also explores his role as part of the IBM SHARE user group, working with others including Hirondo Kuki and Len Harding to review the mathematical routines in its library, critique IBM's early SSP package and, most importantly, to convince IBM of dangerous flaws contained in the initial numerical characteristics of its 360 architecture and to devise an effective fix.

Since 1968 Kahan has been on the faculty of the University of California, Berkeley. He recounts some of the history of its computer science department, including the roles of Abe Taub and Lotfi Zadeh in establishing what were initially two separate departments located in different schools. Kahan discusses his approaches to teaching and the work of his various students, including James Demmel, Peter Tang and Brian T. Smith. Together with his students he produced the widely used fdlibm mathematics library, originally distributed with BSD Unix. He explains the origins of some of his best known work, including the Kahan summation algorithm to correct for rounding errors, and a program he developed to test floating point arithmetic for errors.

Kahan offers a detailed explanation of his work as a consultant with a role in the design of several systems. In the early 1970s he assisted IBM in work on the floating point design of its abortive FS architecture. From 1974-84 he consulted with Hewlett Packard on the mathematical functioning of its calculator range, improving the accuracy and performance of many models and adding the popular integrate and solve functions. During a long and productive relationship with Intel he designed the arithmetic for its i432 and 8087 chips. Kahan explores the history and design philosophy behind the X87 architecture, including its problematic stack behavior. He discusses its impact, the lack of support from compiler writers, and his work on the 80387 chip and on detecting the cause of Intel's notorious 1994 Pentium FDIV bug. Kahan also discusses his work to lobby for improvements in the numerical behavior of several important computer systems, among them the Cray, the CDC 6400, and the Java virtual machine.

In 1989 Kahan won the ACM's Turing Award, for his work in creating the IEEE 754 standard for floating-point computation during the late 1970s and early 1980s. He places this work in context, linking it both to his earlier experiences and to the proliferation of microprocessor designs during the 1970s. Kahan documents the work of others on the committee, including his student Jerome Koonen, and explains the novel features of the new standard. He stresses that the standard is for a programming environment, rather than just a hardware capability. Kahan has also received the IEEE's Emanuel R. Piore award and has been SIAM's John von Neumann lecturer.

HAIGH: Thank you very much for agreeing to take part in the interview. I wonder if you could begin by discussing, in general terms, your early life and perhaps your family background.

KAHAN: I was really very clever before I was born. I chose the right place, the right time, and extremely suitable parents; of course, I'd like to take credit for all of that. Being born in the depths of the Depression may seem like a bad thing. I'll admit that I was too young for World War II. I was in Canada, so I didn't get into the Korean conflict. I was too old for the Vietnam War when I came to the U.S., which was in 1969. I missed many of the risked experiences of people suffered in the generations just around mine.

Being born in Canada was a privilege. Canada was very good to us despite the fact that anti-Semitism was endemic, and racism was endemic in ways that, perhaps, people have forgotten. The position of women was appalling, so a lot of things have changed. My parents were decades ahead of their time. For example, in our house, it was forbidden to use the epithets that were current at the time. We never said *nigger* because my parents would have been offended. We said *Negro*, which was the respectable term at the time. We never said *Chink*. Now, *Jap* and *Kraut* were words that were hard to avoid because of the Second World War. We were forbidden to use the word *shiksa,* which is a Yiddish term for a gentile woman when you want to make a disparaging comment about some Jewish man who has taken up with a gentile woman. We would not use that word. We wouldn't use words like *coon* for Negroes and, of course, we weren't altogether happy if people used words like *kike* for Jews. But what we learned was that, when people use these disparaging epithets, they reveal more about the person who uses them than about the person at whom they're aimed.

My mother had suffered by being dismissed. She was a dress designer, and my father ran a factory and sold the dresses that my mother designed in Canada. They came in time to be very well known in Canada, and earned various awards. My mother had various medals and things like that. She made that tapestry after she retired and a number of others that have been distributed. So I was aware right from the outset that my mother's situation was disadvantaged by her gender and, nowadays, I get offended when people speak disparagingly of women; they're speaking about somebody like my mother or, for that matter, my wife and, ultimately, my granddaughters.

Well, we could have been described as premature anti-fascists. I was reading newspapers when I was about five and a fraction, which means that I was reading newspapers in late 1938. So I knew about the Japanese rape of Nanking, which occurred the previous year, and was still being talked about in the papers. I knew about Hitler's invasion of Czechoslovakia in order to defend the rest of Czechoslovakia from the British. I remember starkly the Molotov-Ribbentrop Pact, because my mother and my aunt were actually members of the Canadian Communist Party—and that split the party. So I was very aware of the split. Those who went along with Stalin and said, "Well, this is the war against Fascism. It's just a war to exhaust the working class." You just can't imagine how they had to turn things around after the Molotov-Ribbontrop Pact. And then, of course, Stalin got his country into trouble in 1941 when the Germans attacked. I remember when Hitler attacked Poland: big, black headlines of war. My father pulled out a picture of his family, and he said, "This is my cousin so-and-so; you'll never see him again."

I mention this because my formative years were spent during World War II. I wasn't in it. I was in Canada. God, I was lucky! But we read the papers. I read the papers. I read the casualty lists, and I think that had an effect upon me unlike the effect that reading the news has now. You can read about things in Iraq and so on, but I don't think that penetrates our consciousness so deeply as events of the Second World War. I mean, if the Second World War had gone a different way, I wouldn't be here, and you would be speaking German. It's as simple as that.

HAIGH: So do you think those early experiences of exposure to the war or to left-wing politics had any influence on the way that you approached your later career?

KAHAN: Oh, yes. You see, it gave me an important insight into the nature of evil and the difficulties in affecting people in mobs. The lesson that I drew more than any other, perhaps, was to avoid crowds: because people will do in crowds where they have a certain amount of anonymity what they would be ashamed to do as individuals. Decent people, people who imagine that they're decent and virtuous, in fact, change when they're in crowds. There's something about the mentality of the mob. So that certainly affected many things about me.

It means, for example, that when Cody says he would surprised that I would get on a committee, boy was he right to be surprised because I hate committees. [W. J. Cody Oral history interview by Thomas Haigh, 3 and 4 August, 2004, Glen Ellyn, Illinois, Society for Industrial and Applied Mathematics, Philadelphia, PA]. You see, the first thing that the committee does is compromise. What does compromise mean in the U.S.? It means everybody is supposed to feel equal pain. It doesn't have anything to do with doing what is right. What makes matters even worse is that you work on a committee and you finally come to some consensus, hard-earned, and then the people who have commissioned the committee say, "Oh, thank you; we're definitely going to follow your recommendations." And then they don't. So, I learned repeatedly that working with a committee is like being with a mob. People will do things on a committee that they would not countenance as individuals if they had to sign just their own name to it. And that means, for example, that in the work I choose to do I have, repeatedly, found that if I can see that somebody else is going to do the work and they're going to do it well enough (maybe they're not going to do it the way I would do it, but they're going to do it well enough) then I can do something else. And that's happened over and over again.

Let me give you a concrete example. Hirondo Kuki turns up in Cody's recollections, and in mine, quite often. Cody doesn't know, apparently, how Kuki got into this racket. What happened was that, in the early '60s, I had found that IBMs library of elementary functions wasn't really all that great. They were inaccurate in spots. They had anomalous properties in spots, like nonmonotonicity. So I set about fixing them, but not very quickly because I was an assistant professor at the time. I had to teach classes. I had responsibilities for advising at the computing center at Toronto. I could fix these things only at a modest rate. And at a SHARE meeting in the early 1960s, I think it was '61 or '62, was Clemens Roothaan, who was at that time a professor at the University of Chicago, with a computing center of his own. He had his own 7094, ultimately. And, I think he is still alive. I know I met him a few years ago, a very few years ago, at Hewlett-Packard. One of his protégés is Bob Worley. I think Bob Worley works at Hewlett-Packard; he brought him in as a consultant. But at that time, Roothaan had his own computing center, and he had his own computer for doing computations in chemistry; Hirondo Kuki was one of his programmers. When Roothaan realized that I was only going to be able to do these things at a trickle rate, he said, "Oh, I've got someone who'll do that." And he set Hirondo Kuki to rewriting the math library for the 7094 and, ultimately, for the 7074 and other machines. And then, on the strength of the very good job that Hirondo did, they got a contract to write the Elementary Transcendental Functional Library for the IBM 360 before any of the rest of us knew it was coming. That's what got Hirondo into this racket. And we'll get back to that later. I know that you'd like to have things a little bit chronological, but once Hirondo was doing it, I didn't have to. I mean, maybe I would have done it a little bit better than Hirondo, but I would hate to have to bet money on that. Hirondo died prematurely of cancer, if memory serves. It was a nasty loss, and I'm glad to see that there are some of us who still remember him. He's going to come up again in this story. So that gives you an idea, then. Aristotle said that we are what we do habitually, and it is my habit that, if I see that something is going to get done by someone else, then they don't need me. I'm off to something else.

Well, my parents valued education very highly and, therefore, they sacrificed a great deal of their comfort to move us to the edge of a small municipality. It subsequently became a suburb of Toronto. It's called Forest Hill Village, surrounded by Toronto. And it had what was reputedly the best school system. That school system made my brother and I what we are. I have no doubt of that. I think I told you before: I'm not a self-made man. That school system did wonders for me. Aside from opening my eyes to so many things, they did what the school system should do in a paramount way: first, to attempt to cultivate leadership. Now, 'leadership' doesn't mean like in John Wayne shouting out, "Follow me, boys!" and rushing off into battle. There are many, many styles of leadership, but the qualities that make for good leadership are not merely the ability to sway other people, but also the good sense to know where people should be led if they must be led. And all of us have to be led in some aspect of our lives, you see. That was the first obligation of the school system. The second obligation of the school system was to enable people to find out what they could do well, which means they have to try a lot of different things. You see, we don't educate people just to stuff facts into their heads, although the facts are the vehicle. The thing we want to accomplish is this combination of leadership and a kind of self-realization that goes beyond the introspective visions of transcendental meditation and other things that are sold around here. It's that people should flourish in our society by doing things that contribute to society's welfare. They should be rewarded for what they do, but they should get the opportunity to do something that they're well suited to do. And they have to find that out. You have to find it out in school, because you find it by trying different things. That Forest Hill school system was really good at that.

My brother and I both got jumped a couple of grades. We both flourished in high school. My brother was a biochemist. He developed, with his second wife, Amante, an extraordinarily potent antibiotic called Primaxin. And, more important, my brother discovered what had, until then, been misdiagnosed as resistance acquired by microbes that had been treated with various antibiotics but not strongly enough. This wasn't the only way in which antibiotics failed. Antibiotics often failed because, having been administered previously to somebody, that person had developed enzymes to metabolize the antibiotic. And then, the enzymes were still present when this antibiotic was reapplied on a subsequent occasion, so the enzymes would decompose the antibiotic before it could get to the bugs. This meant that you had to accompany antibiotics with enzyme inhibitors. Well, that discovery prolonged the life of all sorts of antibiotics, and it prolonged the life of I don't know how many hundreds of thousands of people. So my younger brother, despite the fact that he was a nuisance at times, a pest, he's done more for mankind than all the rest of my family put together.

HAIGH: And what is his name?

KAHAN: Fred, and he lives with his wife Jane in New Jersey. In fact, we saw them a couple of weeks ago, came down for this bar mitzvah.

HAIGH: While many people pronounce your name *Ka-hahn*, you pronounce it more like *Khan*? Is that right?

KAHAN: Yes, my name has some history. My mother came from Moldova; it used to be called Bessarabia. But she was descended from Jews who had been exiled to Spain by the Romans, and then driven out of Spain by the Inquisition, then went to Germany, and then transferred to Russia by Catherine the Great. My mother was a snob and thought she was part of an upper class, despite the fact that they were dirt-poor. They survived the famine that accompanied the First World War only because her grandfather, who had been an estate manager, rendered some service to his lord for which he was granted lands—illegally—because Jews weren't allowed to own land in Russia, at that time. But that didn't matter to the lord, so he was granted that land, and a fragment of that land descended to my mother's father. They were the only Jews in this tiny village that was just a crossroads. But they had enough land to raise vegetables, and had a goat and a cow and chickens. And that was how they survived the famine, which came upon Russia at the end of the First World War, and persisted into 1920 because the Treaty of Versailles did not restore lands occupied by the Germans to Russia. They were Bolsheviks, you see, and they were fighting. And so that particular bit of territory went to Romania which, of course, misgoverned it—which was traditional for Romania. My mother felt that hers was a distinguished family.

My father, on the other hand, came from a very ordinary Jewish family; they had been in Poland a thousand years. Probably their ancestors had converted as part of a conversion of Khazar kingdom. And around 600-odd A.D., they were subsequently displaced by the Tartars and ended up moving through the Magyar territories and were granted the right to live in what, subsequently, became predominantly Jewish towns in Poland and Hungary and Lithuania. So my father said, "What name would you like?" Well, his uncle had come across first, and had been told by the immigration man, at the doors to the Promised Land, so to speak, "Oh—you will never get anywhere with a long, complicated Polish name like that. You need to change it." And so my great-uncle asked what should he change it to. The immigration man said that you Jews are all Levys and Cohens and things like that (which they weren't, but that was what he remembered). Since the *K* sound originated with my great-uncle, because that was what his name began with, he chose *Cohen*.

This is significant because *Cohen* is the name normally attached to Jews who are descended ostensibly from the priestly clan that is descended from Moses' brother Aaron, the Kohanim. They are the vestigial remains of a priestly class whose function in the temples of Jerusalem was elaborate, but their function withered when the temple was destroyed in 70 A.D. So it was an unfortunate choice of name, because we are not in fact descended from the Kohanim. But my father was using his name when my mother agreed to marry him, and he changed it to *Kahan*, because that's what my mother chose in order that it would sound vaguely like the rest of his family. He had had a younger brother who came over, and two sisters. The rest of his family was left at the tender mercies of Germany, alas, except for two sisters who survived Auschwitz.

Auschwitz—"all exaggerated?" I still have one of two aunts alive who can testify differently, from personal experience. They managed to survive so long as they did because they were hidden on a Polish farmer's farm; he pretended they were cousins. They were not related, but out of his soft-heartedness, he pretended they were cousins living on the farm. Then in the winter of 1944, which was terribly severe, there was famine in Poland because the Germans took all the food they could get and didn't care if the Poles starved. A neighbor of this farmer turned him into the Germans, telling them those two women are Jews. That was good for a sack of potatoes. The farmer and his family were shot, and my father's two younger sisters were sent to Auschwitz. They were fairly healthy; they had been working on the farm, so they didn't get into the line for the gas chamber until after the Russians came, and they didn't go into the smoke. I remember the rejoicing in our family when we heard that they had survived. They were the last remnants of what had been a large family on my father's side. On my mother's side, I think the only survivor was a cousin who flew in Sturmoviks. I can explain that later, if it matters to you. But there was a kind of rejoicing when these two women turned up and they had husbands. And when I think of their children and their grandchildren, it's like cocking a snook at Hitler that they survived.

But in any event, all members of my father's father's family have different surnames. Because they were a fractious bunch. And my mother did not diminish the fractiousness of the family. If anything she increased it. So they all had different surnames, and that is why it is very hard to find my relatives. If you find somebody named *Kahan*, if it's not my brother or my sons or his son, they're not related. And there are apparently hundreds of them. Anyway, you asked.

The other thing you asked about was why people call me *Velvel* or *Vel*. Well, my parents should have named me *Wolf*, after my father's grandfather, who was a glazier. We were sensitized to anti-Semitism because the pogroms had driven my mother from Russia via Budapest, Armenia, Hamburg, and Canada. Anti-Semitism was particularly virulent in Poland, and it was one of the reasons why my father left. And so they decided to give their two sons Anglo-Saxon-sounding names, *William* and *Fred*. But of course, around the house, they wanted to call me *Wolf*—except that you can't call a little baby *Wolf* so they called me the diminutive, which is *Velvel* in Yiddish.

Now, I persist in that name because when a kindergarten teacher asked me my name and I said, "Velvel," and she said, "Velvet? Strange name for a little boy." The people who know me call me *Velvel*. If someone yells out, "Hey, Velvel!" on the street, I presume I should know him. If they call out, "William," or "Bill," then I presume that I don't know them that well. In the '60s or '70s, it usually identified someone who wanted consulting for free or an insurance salesman, or something like that. So normally I want people to refer to me simply by my initial, W. On an official document it has to be William with an extra initial. And after a while when people get to know me, they call me *Velvel*, and that includes my students—that is, once they get to a certain point, then they become familiar, so they call me *Velvel* too, although there are a few who still feel that they should call me Professor Kahan, but that is their choice, not mine.

HAIGH: During your time in high school, were you focusing your studies on science and engineering?

KAHAN: Not engineering. I don't think engineering figured in the high school curriculum at all. Science—yes, absolutely. I loved science and math. I didn't like memorizing languages so much, but because I could, I would frequently spend my time in language classes at the back of the room reading this stuff stuck in shelves, which had been used for those classes back in Victorian times, for all I know. So, for example, in Latin class, I would read suitably simplified versions of Horace and Caesar and others. I only took Latin for two years, I think it was, but I got a chance to read a lot of stuff at the back of the class. And with French class it was the same thing. I read *Les Miserables* in a suitably simplified version in French before I read it in English, *Twenty Thousand Leagues Under the Sea* in, again, somewhat-simplified French for schoolchildren. And the teachers left me alone, because if they had a question, I knew what we were supposed to learn in class. And I hated memorizing it, but you do what you have to do. So, I didn't like languages that much, and I despised the way that history was being taught, which is relevant for you. Mr. Langford, who was a decent man and tried his best, had to prepare us for exams. So, we had to know that the American Revolution had 20 causes, not 19 or 21. We had to know how to list them, you see. Oh, that killed history for everybody. If you look at my shelves, you'll see it's not as if I'm not fond of history. I mean, I'm loaded with historical books.

But despite that, I did really well in high school, and I found a number of kindred spirits, of whom the most important was David Gauthier. His father was actually a teacher in the Forest Hill school system. David truly deserved to be called brilliant. He was brilliant in a way that would put the rest of us in the shade. So having him in the same class as I might take-- he was good at everything—really good at everything. It set a certain standard by which I could measure my own efforts. I wasn't going to measure my efforts against the average. I was measuring my average against the best. That meant a lot. He and I were friends. We actually entered math, physics, and chemistry together in college in 1950, but he very soon discovered that that wasn't what he wanted to do and switched to philosophy. I think he's written books on ethics, and I don't know where he is now. I know that he was at Oxford for a while, but I think he's in the United States now, if he's still alive. He was a professor at the University of Toronto for a while, and I was there, too. So there were some remarkable guys at my school.

I took up radio and electronics, generally as a hobby, starting when I was eleven. What impressed me was the fact that you could predict so accurately, relative to the standards of the time, what was going to happen by using mathematics. At that time, it was fairly elementary mathematics. And because I was mechanically adept and still am-- I mean, I repair my car. In fact, it's running a bit rough. I have to replace an oxygen sensor. That's why I can keep these 22-year-old cars alive. I mean, who knows how to fix a French car nowadays over here? Peugeot's not on the American market anymore. It hasn't been for over a decade. So I'd repair my bicycle and repair plumbing and repair appliances around the house.

There's a story whose significance you'll appreciate later. One of our neighbors, just two doors away, was Phyllis Bloom, and she had about finished high school just as I was entering. She taught me how to take a formal square root. Why? Well, in order to make a radio in those days, you have to wind a coil. And the inductance of a coil is proportional to the square of the number of turns. So if you know what inductance you need to go with the capacitance so you can tune the thing to make a crystal set, you're going to have to take the square root to figure out how many turns. I didn't know how to compute a square root. The formal process for taking a square root looks like long division. I don't know if you remember it. You may have been learning it in England.

HAIGH: Let me see, we certainly did long division. I don't remember learning how to find a square root—although we weren't allowed to use calculators until we started on A-level at 16. So up until then, it was pencil and paper and tables all the way.

KAHAN: Well, the way you would take a square root in high school, in my time, was to compute its logarithm, halve the logarithm, and then take the anti-logarithm.

HAIGH: That starts to sound familiar.

KAHAN: Yes, it should sound familiar. And you'd learn about that in the course of learning trigonometry, because you need a square root for some of the calculations, like the area of triangle. You have to take a square root. But, when Phyllis was in school, she had been taught the formal process for taking a square root, and she taught that to me. So I was able to figure out how many turns I needed and all that sort of stuff. And then radios with vacuum tubes used to go bad for so many reasons, principally because the vacuum tubes were not very long-lived, not like transistors. So I would fix our radio and a neighbor's radio. Sometimes I'd have to replace something other than a vacuum tube, and I learned how to use a voltmeter and things like that. I built up a certain interest in radio.

And when I was eleven, we still thought that the war might go on for many years. Now, that was 1944. I remember thinking that the Normandy invasion was a birthday present for me. That was on June 6, and my birthday was June 5. And I thought, "My goodness, they did it just for me." And in a certain sense, they did. They did it for my generation. But progress was slow, and there were many things we didn't know which, had we known them, might have intimated the war wasn't going to last as long as we feared. I mean, we knew nothing about the Manhattan Project. We knew nothing of the decryption of the German ciphers. We were very ill informed about the Russians. We didn't know that the Russians were out-producing the Germans in tanks and in artillery and in aircraft by the end of 1943. Well, maybe, had we known that, I wouldn't have been so convinced that I was going to be involved in this war personally. And because my eyesight started to go very bad, I was no longer able to shoot a rifle. We had a little bit of practice, things like that. I had been a very good marksman until my astigmatism got really bad. I couldn't see the pip at the end of the barrel and line it up with the target simultaneously. My eyes just wouldn't allow that, so I figured I was going to be a navigator and radio operator on a bomber or on a naval vessel or something like that. I felt I was preparing. And the war ended. Hallelujah! But in the meantime, my interest had already been shaped.

HAIGH: It sounds, from your story of electronics, that one of the things you learned from this was that mathematics was something that could actually be useful, which I think many people don't really have a sense of.

KAHAN: Oh, it hadn't ever occurred to me that mathematics wasn't useful. Even if it had been useful only for entertainment the way chess is useful, I still read the chess problem every morning in the paper. They aren't usually very hard. It's just something I do because it keeps a part of my brain alive that wouldn't otherwise be exercised. I was perfectly happy to learn about mathematics because I thought it was intrinsically interesting. But, I think I differed from others in this crucial respect: to me, mathematics then, and even more so now, has meaning beyond the symbols that you write out to express it. The symbols, you see, are written linearly. I mean, you've got to write words, sentences and so on, and the same thing is true in mathematics, although there are branches of mathematics where you draw graphs and other diagrams. The fact remains that a proof is a linear concatenation of symbols. If you can't do it that way, it isn't a proof.

But, even in high school, the mathematical relations had other meanings. The whole point of mathematics seemed to me then, and seems very strongly to me now that I've become a professional mathematician, is that it acquires many different meanings depending on how you look at it because of its abstractness. What abstraction means is that you have found something in common amongst otherwise disparate objects. It's trivial to say, "Well, three is the abstraction held in common among all sets with three objects in them." But it's more than that because, you see, when you say three, if you say it in a context, in that context it means something about some set with three things. Then you realize that the meaning that it has in one context casts a light on other contexts. And the way you solve a mathematical problem is, very often, to see from a different vantage point what that problem is about. A vantage point that, perhaps, others haven't taken, in which case you may end up solving a problem they haven't solved before. But it doesn't matter. The important thing is that when you look at these things from different vantage points, it's like getting a certain roundness of the subject. And then it has a reality. It becomes just as concrete as anything else in the world, so you may guess that I'm a Platonist.

I think I appreciated early, without being able to verbalize it until later years, that the marvel is that you can so well understand, and even, sometimes, control things, by manipulating not the things, but instead, the symbols that stand for them, after a fashion…not perfectly, perhaps. And if I stop there, I'd be a physicist. But then you look at the symbols, and you say, "My goodness, they're things, too." And when you start applying the same idea to the symbols in things—and then you do that recursively—then you're a mathematician. And I don't mind sliding up and down the scale but for the grace of God (as I'll explain later), I might have been a physicist. So, I didn't have this sense explicitly when I was a teenager, but I know now that that's what made it so interesting. And you see, I could make it concrete because, with things that are electrical, even with antennas, you can predict so much about what they will do by performing a computation. Well, nowadays, we're computing numerically because we allow very irregular shapes. We have much more general situations to deal with. But at that time, the shapes were simple. The setups were simple. We ignored many parasitic effects. In that case, too, within one percent or so, you could predict what was going to happen, and that appealed to me very greatly. Now, you can predict also mechanical things mathematically, but the computations are heavier.

Of course, at that time, people despaired of predicting anything about biological systems. And, in fact, I think there probably are people who do still feel the way the chairman of the biology department felt when my brother was working on his thesis. He was solving differential equations that would make it possible to correct for unfortunate effects that occur when you wish to study the influence of cell walls on metabolism. What you do is you put in various nutrients, and you tag them with radioactive isotopes. Then, certainly these materials find their way into the cell. In the meantime, they're decaying radioactively. Finally, you try to fetch the cells out, and you freeze them or do something to them. Then you mash them, and you put them in a centrifuge. And you try to find out how much of the stuff got into the cell. In the meantime, tick, tick, tick, the radioactivity is decaying. My brother solved some differential equations so that you could compensate for the amount of time it had taken you to do this preparation. That was ultimately adopted as a standard technique. He and his advisor wrote the stuff up. But in the meantime, the chairman of the department had told my brother, "Whatever you learn about life with mathematics is almost certainly wrong." Now, he could have meant that my brother was mathematically inept, but he really meant that life was intrinsically refractory to mathematical thought. We don't believe that now, do we?

HAIGH: Not in science.

KAHAN: Not in science, that's right.

HAIGH: I wouldn't venture to speculate for the population as a whole.

KAHAN: No, that's right.[1]

[1] HAIGH: Let me ask you a follow-up question on all the things you said there, then, before we move back to your career. So, you've, I think in some ways, contrasted the use of abstraction made in physics with the way that you would see things as a professional mathematician. Now, how about computer science? Herbert Simon formulated the idea of computer science as a science of the artificial.

KAHAN: Okay, first let's distinguish computer science from computer engineering. In engineering, you've got to get things to work, and if they don't work, you have to fix them by hook or by crook. So, my colleagues in the electrical engineering and computer science department have a marvelous out. If what they're doing doesn't work well, it's science, and you have to understand it. But if it does work well, then it's engineering. The difference between computer science and engineering, then, has to do with the difference between understanding and doing when building things. There are many things we build. And what's more, when I write software, I've got to tell you there are occasions when it works a hell of a lot better than I can explain. That's engineering. You've just got to try to understand it even though you'll never prove everything mathematically. But when we're thinking science, we really want to understand things. And we want to form that picture I described where, somehow as you look around the subject, it all seems to cohere; you feel that you understand it in the same way as you understand the streets near where you live. But computer science is a branch of applied mathematics, preoccupied with the cost of computing. The cost can mean the cost in time. Actually, everything—everything, without exception—can be measured in terms of time (but if you want to dispute that, we can dispute it later). You may ask can human relations be measured in terms of time, and yes they can, in a perverse sort of way. But let's not deal with that right now. So we're concerned, as computer scientists, with how much memory and how much time will it take. How long will it take you to write the program? How long will it take you to debug it if you ever do?

These are all cost questions, whereas the applied mathematician, when he is merely an applied mathematician, is interested in whether a mathematical relationship is adequate approximation. Mathematicians aren't taught to model, you see. Physicists are, but mathematicians aren't. So the physicist's job is to find a mathematical model, which is, on the one hand, a good enough approximation and then, on the other hand, tractable. If he approximates too well, it'll intractable; if he makes it too easy to deal with mathematically, it may just leave too much out. So the physicist comes up with the model, and the applied mathematician wonders whether that model resembles anything that's been seen before. So he's got this problem, and he's looking for a solution. And this solution may involve existence, or it may involve qualitative behavior, and so on. But it's all in an idealized world in which everything is done exactly even when you approximate, right? That sounds paradoxical, but it's like a Taylor series. A Taylor series represents the function perfectly, and then you decide to take only so many terms. Well, that's' the applied mathematician in me. A pure mathematician, on the other hand, is a man with a solution looking for its problem. Sometimes I can be a pure mathematician, too. And the distinction between pure and applied doesn't seem to me to be terribly enlightening, because we flip back and forth. At least, all the guys I know who are interesting flip back and forth. But I've got to tell you, there are some people who think of themselves as pure mathematicians and, unfortunately, they're snobs. They look down on what they regard as applied mathematicians—and that is a serious personality defect.

But this sense of the power of abstraction figures importantly in computer science, where the abstractions have to be reduced to language, or else we can't use them. For an applied mathematician, he can have a picture in his mind, which may be geometrical or it may be a combinatorial, configurational. You can have all sorts of visions in your mind, which are not necessarily in close correspondence to linear language. And different people think better in some of those ways. So people with combinatorial mentalities (and I'm not one of those) have minds that are marvels to behold, because they obviously don't think in simply linear terms. For them, it's a major problem of translation to translate their combinatorial structure into linear terms. And on a continuum, then, I like to think of things evolving, of shapes transforming themselves continuously. So, partial differential equations, ordinary differential equations, that's the world I prefer to live in, if I have my druthers. And it's a little bit easier to explain these things in linearized terms thanks to the calculus.

Then there's also the world of geometry, which I love. I'm not really all that good at it, but I love it. I mean, it's a marvelous way of thinking. So the geometer thinks of spaces and configurations in those spaces, and that affects classical analysis, because a lot of the best understanding of classical analysis and differential equations and functions basis comes about because you think about it geometrically. But in computing, we have to reduce it all to linear terms, you see, and language is deceitful.

HAIGH: But obviously you've got the problem that you have to express something in terms of a computer program, but you can't reason well about what it does, in many cases, other than by running it. So it introduces a certain element of experimentalness.

KAHAN: Oh, yes, and I'm not ashamed to run experiments. But no, I hope you are not right when you say you can't reason about it except by running it.

HAIGH: In the general case.

KAHAN: In the more general case, but then that's an unfortunate program. We really have to try very hard to write our programs so that we can reason about them in a way better than single-stepping through them. You see, the machine is going to do that several billion times as fast as we do, so if all we can do is single-step through it, forget about it. That's what I told you. The machine is going to do it a hell of a lot better than I am, so I'm not going to do that. I've got to find a way to devise the program, perhaps, to ensure that there is certain invariance. I've got to have conceptual structures which are better, more potent, more penetrating than the text of the program, or else the program may turn out to be--

I'll give you an example, because I've looked and looked and looked to find this program, and I don't know where the hell it is. I wrote a program in the early 1960s to do numerical quadrature adaptively. Now, what the word adaptively means is essentially this: a numerical quadrature is an attempt to evaluate an integral. We replace the integral by an average of discreetly sampled values. If you divide the integral by the widths of the domain of integration, it's an average. So, integration is essentially averaging. That's the way I want you to think about it, for the time being. And numerical quadrature averages discreet samples instead of averaging continuously. But of course, you want to compute averages that are very close approximations to the integral. One way of doing that is to compute and average for a set of samples. That's your initial average. And then you subdivide, scatter samples more densely, compute a new average, look at the difference between the two averages, and then, if you've done it systematically enough, it'll give you some insight into the error. The difference between the averages of the discreet samples and the averages of the continuous function, it'll give you an idea about that, and then you see if the error is to big, it'll say, "Oops, I better subdivide further." You see, typically, we use a formula for averaging which has the property that, if you double the sample density of a uniform sample, you reduce the error by a factor of 16 asymptotically, as density increases and assuming the function is smooth enough, and things like that.

Okay, so this is all very technical at first sight, but you see, adaptivity means that when you find that the error for averages in some subregion is intolerably big, what you have to do is subdivide in that region. Whereas, in some other region, a sparse sampling may be good enough. That function may be so nearly straight--

[Tape 1, Side B]

KAHAN: So in adaptive quadrature, you realize that the density of sampling may have to vary across the region of integration. The way you vary it is to obtain these error estimates by doing repeated averaging. You average course, average answer, compare. And if the comparison indicates to you there's too big a disparity, you infer that the error is too big and you subdivide further. Doesn't that sound recursive? Of course. I mean, you take an interval and subinterval, say. You do something to it, and then you do something to sub-subintervals. And then if you don't like that, you break the sub-subintervals into sub-sub-subintervals, and so on. I mean, it's so obviously recursive, and that's the way it was done in ALGOL.

HAIGH: So the base case is an interval where the divergence is acceptable?

KAHAN: Ultimately, if you subdivide enough, the averages will agree well enough, and then you'll accept the average. And you'll average the averages here with averages over there. You have to weight them according to the width of the integrals. And sooner or later, you get to the point where you've got an average of everything, more or less uniformly weighted, even though the samples are very much denser here than there. And therefore, they're weighting individually is less, because you're weighting them by an amount proportional to the gap between samples. Okay, the first schemes that did this were recursive. I think it was Bill McKeeman at Stanford who wrote one of the early ALGOL programs to do adaptive quadrature recursively. But Fortran is not recursive, or at least it wasn't then. I mean, now everything's recursive, but Fortran wasn't. And there I was, stuck. I'm working on a 7090, in Fortran and I can't call it recursively. And if I did, bad things would happen. You see, when you call a function recursively, what do you do? You push a pile of stuff on the stack—not just the arguments, but also return address and who knows what other stuff, like scratch variables. Well, if you have to subdivide a lot, you're going to build up an awfully big heap of things on the stack—and we only had to 32,000 words on a 7090. So recursive was not the way to go if you had an integral which, when evaluated adaptively, had an extremely wide range of densities. I remember writing a beautiful program, completely non-recursive, to keep track of sampling densities in a 36-bit integer. You look at the bits in that integer, and you could decode that into telling you how wide the subintervals were that you were working on. Roughly speaking, it meant every time you subdivided you double the integer and stick a one in there. And then, when you looked at the other half of the subdivision, you'd knock the last bit off. When you combined the things, if they combined correctly, you divide by two, roughly. That's the way in which it was encoded. I can't find the program. I've looked and looked and looked because it was such a gem.

And I think it was submitted to SHARE, and it would be a good example of what I've tried to explain to you about linear thought. Translating the conceptualization of that program into Fortran, linearizing it, you see—that's what took all the energy and the effort and the care and the craftsmanship. But when you read the program, you can't infer the structure by just reading the program; there has to be some other kind of mental structure besides just the linear one. And that applies also to most mathematics. There is some mathematics that is just algebra. And some of the automated algebra systems are good at that. But the reason that we do things as humans the way we do—if we succeed—it's because we have these extra notions. And I guess it's a perverse connection.

You've probably read that, in various jurisdictions, people are resolved to put more money into math and science education, because we are suffering already a severe shortage of people who are competent in mathematical technologies. And the shortage is obviously going to get a hell of a lot worse, and we can't depend upon foreign countries to make up the deficit. For one thing, the United States has made itself unpopular in many places. There are people now who would feel a loss of self-respect to come here. Ten years ago, the same people would have thought coming to the United States was a good idea. Well, so we're going to pour money into science and mathematics. It's going to be futile. It isn't going to work, and the reason it's not going to work is because, in science and mathematics, the use of language is different than the colloquial use. And you have to be sensitive to that. You have to be sensitive to a precise use of language in a world where imprecision is the norm. How will students learn to use language precisely? Well, you can try to teach them grammar. Grammar has fallen out of favor in schools, nowadays. Maybe you could stick grammar back, but grammar is very dull. It's just a lot of rules. Many of them are capricious and arbitrary. In order to use language well, you need examples. Some of those examples will rub off on you. That's literature, so English literature has to be taught. But how much literature can you read without some knowledge of history, because the history provides the context in which the literature makes sense, with rare exceptions. You can have some very fanciful literature, and science fiction, for example, tries to do that. But if you try to read science fiction and you don't have some knowledge of history, you're not going to understand what the analogies are about.

HAIGH: Well, science fiction is all about history.

KAHAN: I think so.

HAIGH: In any science fiction novel, there's a point where the world the story is set in departs from our own. In some cases, that point is in the future to the writer, or in some cases it's in the past. But it's all about thinking how history might progress from where we are at the moment or about how history might have progressed from a given point in the past.

KAHAN: I only mention science fiction because there are people who imagine that, somehow, in science fiction you don't have to know history. But as you pointed out, it's wrong. It's utterly wrong, so you've got to learn some history.

Well, so look at that. We should be investing in English teachers, teachers in English literature, teachers of history. Why should they teach poetry? Well, the reason a scientist has to know something about poetry is because there are things that language doesn't quite do if you insist on doing it in prose. There's a certain kind of poetry in mathematics and in physics. There are certain patterns that have an aesthetic appeal, and we tend to them naturally when we choose an apt notation. It's very much like poetry in sentiment. Choosing the right words, Mark Twain put it, "The right word is to almost the right word as lightning is to the lightning bug." And in poetry, part of the art comes in choosing just the right word, sometimes with ambiguity chosen with the malice of forethought, benign

KAHAN: In the summer of 1951, I got a summer job through the agency of a school acquaintance, working for a company called Stark Electrical Equipment. They produced multimeters; in fact they were the only producers of voltmeters and ammeters in Canada. I still have one of theirs. They had secured a contract to repair equipment for the RCAF. Some of this equipment was the Huff-Duff equipment that was used to locate submarines that were incautious enough to broadcast this very short, squirted message back to home base. That equipment was obsolete by the time we were repairing it. I think that was a futile endeavor. But at least I got to understand a bit about how that equipment worked. However, part of the contract also included the repair of measuring equipment. *AVO* is the name of the British manufacturer of a combination volt-ohm ammeter. Very good quality machines, but of course, many of them had been damaged by other use. I and a Polish immigrant working there—they were employing immigrants, legal immigrants, but because they spoke English poorly, if at all—they were hired at very low wages, which is what made this whole contractor economical for Stark Electric Equipment. There was one guy, a Polish fellow; he understood something about electricity because he had taken it in college. He knew Ohm's Law and I knew Ohm's Law, and nobody else in that entire department did—not the foreman nor anyone else. They would get their instructions from someone to un-solder these parts and replace them with new ones or whatever, but they had no capability whatever of diagnosing anything. This was important in the repair of meters, because the meters had to be accurate to within a few percent. So that took some skill at figuring out what it is you were supposed to do. And so when they found out two things, one was that I understood Ohm's Law, and the other was that I was articulate and willing, my job was to prepare what we have now called "fault trees." Namely, if an instrument comes in, what are the things you measure? Record the measurements, now look at these measurements in order to see

---

ambiguity, or even creative ambiguity. And that's why we want people to learn something about poetry, so that they're going to appreciate that language isn't just about prose.

Why should students learn about music? Well, for some people, music speaks to us in other language can't, so it sensitizes us to the notion that there are some things that can be communicated not so well in language as in music. I don't mean rock music, because that's just raucous to my ear. Or maybe I'm just too sensitive to nuances that don't really deserve to be there. My older son certainly thinks so. Although he's a lawyer, he works from time to time in a rock group. That's his night job. His day job is a lawyer. Well, I don't see anything in rock music but, to me, Mozart speaks volumes. The sense of humor is just bursting out of his music, except maybe for Requiem and a couple of other little things, so you get some sense of what he was saying in a way that would be either trite or impossible in ordinary linear language. And people have to know that there are these ways of communicating things. The same is true about fine arts. You have to know that such modes of thought exist in order that a little bit of them will rub off on you in whatever mundane enterprise you're going to attack later. Because if all you do is think along narrow channels without this enrichment, there are problems you will not be able to solve. At worst, there are problems you're going to make worse for the rest of us. You can't enhance the production of scientists and engineers and mathematicians simply by spending money to hire more teachers of math and science. You need the others, too.

But, alas, our politicians appear (almost all of them) to have no sense of this. They're very crass. I mention this because you started by asking me something about the influence on computer science of being a mathematician and what is computer science. And I said, "It's a preoccupation with costs and so on," and that makes it seem terribly dry and businesslike. But a life spent thinking in narrow channels must be a pretty pitiful life. In order to think in broader ways, young people have to be exposed to ideas which may occur to them later on the road. But there is a viscosity in the human mind that limits the rate at which ideas can penetrate, and there are some ideas that would never penetrate, unless you were asked in school to think about them. As one of my friends, Beresford Parlett has said, there are families who, for generations, have never considered the possibility that there are some things worth thinking deeply about.

which conditions are satisfied or dissatisfied, and then follow your way through the fault tree until finally it tells you to replace this. Or at least to replace this first, because sometimes you aren't quite sure what is defective, so you replace one thing in the hopes that it was defective and, if not, you will run your way through the fault tree again and you will discover something else. And so the idea is to replace the cheapest thing if that is the only thing you can do; but usually my fault tree would simply point to the culprit. And this enormously sped up their operations because, previously, what they had been doing was replacing everything that looked slightly dubious. And slightly dubious, well, how can you tell? I also got to repair the moving-coil part of the meters from time to time, although that was done in a separate building. Every now and then I would get an instrument, which they thought they had repaired or else thought didn't need repair but actually did need repair, of the moving coil. Have you actually seen these meters with the needle that swings back and forth?

HAIGH: Yes.

KAHAN: And you may not know exactly how they operate--

HAIGH: I have one on my tape recorder.

KAHAN: Yes, okay. Now the question is, how do these things work? I won't go into that now, except to say that my eyesight was good enough and my hand was steady enough that I could repair these things. Well, what drives that pointer is a small coil on a spindled axis with very sharp bearings and that coil moves in a magnetic field. And without going into details because I understood that and knew what had to be done and I had a steady hand and a good eye, I could fix those things, too. So actually they made a good deal of money out of me. Once they had this fault tree and duplicated it for everyone-- oh, and the other thing about the fault tree was, it wasn't in English. It was little diagrams with various symbols for, you know, resistor or whatever, and once people had learned these things it didn't matter whether they read English or not. The foreman was delighted and, I think, when I left they were sad to see me go.

But in the meantime, I had met a number of interesting people and we would chat over lunch. One of them was a Pole whose family was extremely well to do, so well to do that they owned an airplane. He was an officer in the Polish Air Force, and when his aircraft was shot down by the Germans, he managed to survive. He took his family's plane and escaped to Sweden and then to Britain, and he joined the RAF. He was one of a contingent of Poles like that who were desperately needed during the Battle of Britain and who earned a reputation for reckless foolhardiness in their determination to shoot down Germans. But this particular guy, his friends told me, was demoted because of too many hard landings. Apparently he had broken too many undercarriages in hard landings, and they demoted him and made him part of a bomber crew. He found out first that I was a Jew, and he said, "You know, we don't like Jews very much in Poland." And then he found out that I was not so virulently anti-communist as he, and so one day he threatened me. He said one day you are going to walk past a dark alley, but you aren't going to get all the way past it. Interesting fellow.

I also met a German who had come back to Canada because the two best years of his life were spent in a Canadian internment camp for German prisoners. This was a guy who had served as an anti-aircraft gunnery officer on the Scharnhorst when it was sunk. There were only about 40 survivors and, in British accounts, they said they picked up 39 or 40 ratings. This guy pretended not to be an officer in order not to be interrogated, and so he was interned with ordinary seaman. And I think I have described for you where they were interned, about 150 miles northwest of

Toronto in what had been—and subsequently became again—resort country for people who wanted to escape the heat in the summer. Beautiful country—my favorite country. And he decided that if that was Canada was like, he would like to come back; so he ended up working in this place, too. He didn't know much about Ohm's Law but he was a quick study. He knew a lot about gunnery. His hobby had been radio, but radio as an amateur. He wanted to talk over the radio and thought that when he joined the German Navy that he would see the world as a radio operator. But they had lots of radio operators, so they made him a gunnery officer. And he taught me about naval artillery, the Magnus effect, all kinds of marvelously interesting things, as we would chat over lunch.

Well, that experience prepared me—more than prepared me—for a job that I did later as an undergraduate student in the physics department in repairing instruments, because repairing these moving coil instruments was a really very expensive business if you sent them out. I don't think the physics department could have afforded to repair those meters if they had to send them out. It would cost more to fix than to replace. And then you can't replace it with the same kind so the desks and the benches would look different. They would have different meters. And therefore they would need different instructions. The fact that I could repair them was a plus. I couldn't repair every one, there was some that were just blitzed beyond repair, but I could use salvage parts. So that was how I got that job and it was really interesting work, because that was how I found out about the professor who ran that lab and who was my supervisor. He had helped design vacuum tubes for proximity fuses. It was Anderson. And he had lots of fascinating stories to tell me, too.

HAIGH: Let's return then to your own educational progress. So you've talked about your experiences in high school, about your electronics hobbies, and about your belief that the war would go on for a long time.

KAHAN: Well, the belief ended at 12. It was early in high school.

HAIGH: As you were approaching your graduation from high school, how did you think about your own future and begin planning for what came next?

KAHAN: Well, there was this girl I was very fond of, and we were discussing plans for the future. I said, "Well, it looks as though I'm going to have to take mathematics, because I've got this scholarship." Did I have a scholarship then? I'm not sure. I think I must have because originally I planned to be an electrical engineer. It seemed so natural, and it looked like a good way to go. I mean, at that time, television was a big thing. I could repair television sets. Maybe I could even design television sets, who knew? But this scholarship was tenable only in arts. It wasn't tenable in a professional school, in engineering, in medicine. Those were professional schools. I had to take art, so the government was going to pay me. I thought, well, they might as well pay me to do something hard. So, I thought, I'll take math, which seemed to me to be the hardest subject. And, as I explained to this young lady when I was trying to tell her what I had in mind, she said, "Well, with a degree in math, the possibilities are…to be an actuary," which I described at that time as a fate worse than death. Or, I could become a professor. The prospect seemed to me extremely unlikely. I didn't think I even had an inclination. Although, ironically, my nickname when I was 12 and 13 was "professor" because, as you are discovering, you can't get a short answer out of me. If you get an explanation, it's going to go on for a while. Of course, other people could see that. I couldn't see that in myself. So, I figured I'd end up as a high-school teacher of mathematics, and I said, "Well, you know, high-school teachers of mathematics only get a certain seniority. They have a certain income." And this is what I described to this young lady, and she said, “No, no. Don’t worry. You’re not going to end up like that.” She was right. Of course, we’ve been married for almost 51 years. But I hadn’t planned to go the way I went, but I started with math, physics, and chemistry in my first year.

If I distinguished myself, it was mainly because, as a descendent of peasants, I didn’t know you weren’t supposed to ask professors questions. And so, I’d raise my hand and ask questions. I think my professors were surprised, and my classmates apparently knew better—I don’t know how. But they were surprised. In the second year, sophomore year, we had a professor whose name was Burke. He would come in, face the blackboard, and begin, “Last day, we were considering…” and he’d start writing notes. And his hand was very legible, but quick. And he would dictate his notes as he wrote them on the board, fill up the boards. It was all calculated to take an hour, and then he would leave. And occasionally there would be parts of his notes that seemed to me to be not as clear as I would have liked, so I would ask a question. Because his back was to us, I would have to say, “Professor Burke,” and he’d turn around. I think that he actually enjoyed the questions. The thought that somebody actually cared what the stuff meant seems, to me, to have surprised and amused him—even though his face did not betray a lot. He answered the questions. Unfortunately, he died the following summer, and it was suspected that the shock of answering questions had something to do with his heart attack. (I don’t think so, but that was the word that went around.) By the fourth year, asking questions was relatively common in math, physics, and chemistry. There was a sea change, and the relationships between students and professors, I think, improved.

But, looking ahead, after I left the University of Toronto for a couple of years in Cambridge, and I came back as an assistant professor. Do you think I could get students to ask questions? No. I would make intentional mistakes, hoping that somebody would ask one. No, they just got into their notes. The joke was, how do you tell the difference between a first-year class and a fourth-year class? First-year class, the professor says, “Good morning, class,” and the students say, “Good morning, sir.” In the fourth-year class, the professor says, “Good morning, class,” and the students write it down. I struggled and struggled and still struggle to get students to ask questions…but I lose the struggle all the time.

Anyway, I had started math, physics, and chemistry, and I found a profound distaste for chemistry. It was a messy subject, somewhat stinky. But worse, in the lab class, we were supposed to measure the concentration of something or other, and in order to get a better estimate, we were to separate the fluid into two samples, approximately equal, measure the concentration in one, measure the concentration in the other, and then average them in order to reduce the overall error. Well, unfortunately, I hadn’t stirred well enough, so it became perfectly clear that the concentrations that I measured in the two samples were rather different. So, I still averaged them with the appropriate weight for the amount of material in order to come up with an estimate of what the original concentration should have been had I stirred it thoroughly. Then I wrote this up carefully in my lab report, and I was marked down severely. And I thought, if that’s the sort of martinets that chemists turn into, I want none of it. So, I chose math and physics.

HAIGH: So, at that point, did the University of Toronto have the American-style system with semesters and elective courses?

KAHAN: No. You committed yourself for a year. Classes, some of them were huge. There were 150 people in the first class I took at college, which was a class in calculus taught by the then-chairman of the mathematics department, Sam Beatty. He was a fairly elderly gentleman then.

He came in. And the first words he said to us were, “Most of you are going to flunk, and it’s a good thing.” And, that happened: 27 of the 150 graduated in math, physics, or chemistry. The others went elsewhere.

Well, the market at that time for mathematicians, not to mention physicists and chemists, was not all that great in 1950. Sputnik hadn’t gone up yet, and I didn’t know about computers. I didn’t imagine that they existed at that time. Of course, they did exist, but I didn’t know that. Well, at the end of the second year, I would have to choose between mathematics and physics. I’d already rejected actuarial science because it was insufferably dull. You ask about numerical analysis in your schedule there. We were acquainted with numerical analysis in our first year by a course in actuarial science, which included interpolation and numerical quadrature and other interesting things. Except the instructor and the text (I still have the text), they made this subject unimaginably dull. I sat at the back of the class and found a volume of Newton’s *Principia* in Latin, and whiled away my hours reading that. It would never have dawned on me in a million years that I was going to become a numerical analyst. It just would never have occurred to me.

We had a number of physics classes. One was an optics lab. And a high-school chum and I had gone through much of high school together. We were kindred spirits in many ways, and we were lab partners. And the first thing we used to do was repair the equipment. The second thing we did was we’d calibrate it, and then we ran the experiment. Needless to say, our results would come out rather better than the results of people who hadn’t taken these precautions. And this came to the notice of the professor who ran the lab and, one Friday afternoon at 3:30, he cornered me in the hall. He said, “Kahan, you’ve really got to choose physics.” He knew I had to choose physics or math if I was going to continue. He said, “All the world’s greatest scientists have been physicists.” Well, a couple of things went through my mind. Was Pasteur a physicist? I didn’t think so. Was Darwin a physicist? I didn’t think so. As those thoughts went through my mind, I got a whiff of his breath. And what went through my mind then was, If being a physicist means being drunk at 3:30 on a Friday afternoon, I want none of it. And I chose mathematics. That’s the way these great math choices come about.

HAIGH: Have you found mathematicians to be more sober on the whole?

KAHAN: Of course not; I have no idea whether they are or not. You ask me what influences-- I’ve got to tell you that I used to think that people made their choices intelligently. I used to think that the people who got out of Europe before Hitler took it, had been prescient, wise, farsighted. They were just lucky, and I was lucky. I’m not a self-made man. I tell you, being born at the right time to the right parents to go to the right school. God, that was luck. It wasn’t done. Oh, right, you can conduct yourself in such a way as to deserve luck, but a great deal of what happens is just that. It’s luck.

And that was what sent me into math. And in a way that was a good thing, because although I’d taken a good number of physics courses and continued to, math prepared me better than physics would have for the things that I ended up doing. I might have ended up doing the same things anyway, for all I know, but I think my math background helped a lot. For one thing, I had courses in both my third and fourth years from H. S. M. Coxeter. He died only a few years ago. He was a 19th-century geometer living in the 20th century. And things he did were so beautiful and so elegant that, although I had no plans to be a geometer, I treasured his courses. They set a certain standard for me. When I told you that people have to have these other notions than just the linear notion, you can’t just memorize the formulas. You can’t just follow the algebraic rules. There has to be something else that gives the subject a kind of coherence and meaning, and I think I learned that from Coxeter's courses more than from any other. If you could just see it in a certain way, and Coxeter could see things in four dimensions as I could not. He obviously had a vision in his mind about the way things were, and just a little bit of it rubbed off on me. But that little was priceless.

So still when I do things, I ask myself afterwards, now, is this really a neat and understandable way of doing it? Have I explained it, first to myself and then to others so that I can see that this is the world as it is because it cannot be any other way? But it's still an interesting world because you can look around it. You can see, yes, it all hangs together. And I took courses from Abraham Robinson. Now, Abraham Robinson is famous for non-standard analysis. This is a form of mathematical analysis in which you adjoin to the real number system a set of things that I'll just call infinitesimals. That's close enough. He really wanted to be a logician, but he was teaching us applied math courses. It was things about fluids and a little about differential geometry, necessary to understand elasticity and tensors, things like that, which was not his natural affair. But he had a strange way of thinking, and a little bit of that rubbed off on me and so on and so forth. It was in the math classes that I met the people with the strangest mentalities. They seemed strange and, yet, as I came to understand them, they looked less and less strange. As I came to appreciate that, it's a point of view. And I'm grateful for that.

HAIGH: So the University of Toronto would have at that point been seen as the most prestigious in Canada?

KAHAN: Oh, yes.

HAIGH: And the mathematics program would have been the leading one in the country?

KAHAN: It was at the time, but it lost its preeminence for a God-awful reason. I mentioned that the chairman of the department was Sam Beatty, and he continued as chairman until he was promoted upstairs in my third year. Sam had a number of unfortunate prejudices. He thought that mathematics was classical analysis, the kind of algebra that he understood, which was just a sliver of it, and geometry. In consequence, he disparaged applied mathematics and, therefore, the University of Toronto lost eminent applied mathematicians. [John L.] Synge, he was there for a while, [A. F.] Stevenson, Infeld.

Now, the Infeld case will illustrate the point all too well. Leopold Infeld was a Pole, a Polish Jew who had escaped and who worked, among other things, on relativity and even had some contact with Einstein. And Sam Beatty had no use for him. He was invited to Poland to see whether things could be put back together in Poland, things mathematical. That would have been 1948 or '49, just before I came in as an undergraduate in 1950. Poland had a flourishing mathematical culture, of course, many of them Jews. That was all the more remarkable because, you see, in Poland after 1919, when Poland was established by the Treaty of Versailles, Jews were not allowed to go to school beyond, I think it was grade six. They weren't allowed in most professions. They certainly weren't allowed in universities, but they managed to get in there one way or another.[2] So the invitation to Infeld was an attempt to undo what the Germans had done

[2] KAHAN: And the same sort of thing happened in Hungary when the Iron Guard-- it was Admiral Horthy. He was a fascist in Hungary. The same kinds of things happened. So my father, who would have liked to have been an architect, could not go to school long enough to be an architect, and instead he had to take one of the vocations open to Jews. He became a tailor. Even at that, he was the first member of his family to have what would be called a trade. *Trade*'s a funny word. It really means "a craft." Previous to that, his grandfather had been a glazier, but his family all owned inns or ran inns. They're not allowed to own land, so they ran inns. They were peddlers and stuff like that, but my father was a tailor. So he actually earned a living, you see, instead of getting it by buying and

in killing off the Polish intelligentsia. I mean, it was systematic Nazi policy. Schauder, for example, comes to mind as one of the Jewish mathematicians killed off. Schauder is famous for the Schauder Fixed-Point theorem. Well, while Infeld was in Poland, a politician in the Canadian Parliament, Colonel Drew, then leader of the Progressive Conservative Party (about which there was absolutely nothing progressive) asked the government how they could allow a nuclear scientist to go behind the Iron Curtain. The nuclear scientist was Infeld, who wasn't a nuclear scientist. He was an applied mathematician who had worked on relativity. He knew nothing about nuclear physics. Beatty could have said that but didn't. He just kept mum. So there was this furor in the papers, you know, a tempest in a teapot. And Infeld felt that he was being betrayed by his colleagues in the math department who knew perfectly well what I just said—that he was, A, not a nuclear scientist; and B, had no intention whatever of betraying Canada, which had given him shelter. But things went from bad to worse, and Infeld decided not to come back. That was Sam Beatty's doing because, as chairman, it would have taken only a word from him. And he just didn't say it.

Anyway, he made life uncomfortable for a number of people. Sam Beatty didn't like my algebra professor. That was Ralph G. Stanton. Stanton was a very peculiar fellow. I didn't know about homosexuality then. I didn't learn about homosexuality until I got well into my 20s, but Stanton was. And I think that Beatty despised him for that and therefore didn't promote him. So when the then-minister of education in the province of Ontario found himself with money on his hands, he endowed his former school in Waterloo, Ontario, about 70 miles northwest of Toronto. He endowed them with a lot of money to set up what amounted to a faculty of mathematics. Now, this would have been in the late 1960s, and Stanton had already left Toronto for Waterloo because he had not been promoted.

And Beatty's replacement, I. R. Pounder, was a Christian gentleman in the best sense of the word. He felt the only fair thing to do was to promote people in alphabetical order, and Stanton was late in the alphabet. Tutte was late in the alphabet, and so they did not get promoted the first year around. So they decided to go off to Waterloo. W. T. Tutte, an Englishman— you should know that name, because Bill Tutte had been recruited from Cambridge as an undergraduate to work on the decryption of German ciphers.[3] Then he came to the University of Toronto, where

---

selling. After the war, the Poles attempted to undo what Hitler had done. I don't know if you have seen, as I have, photographs of Warsaw after the Germans left. Nothing is standing higher than a bedpost, and the Poles decided they wanted to rebuild at least the façade of their main streets. They couldn't rebuild everything, but they wanted streets to look something like they had looked. And so, they sent out a call to Poles around the world for photographs. So suddenly, my father became an ex-Pole citizen. Jews had been denied citizenship before the war. Actually, Polish anti-Semitism lives on, but that's a story for another day.

[3] KAHAN: Through a mistake on the part of the Germans, they sent a message twice using the Lorentz machine. Do you know what the Lorentz machine was? Fish was the British code word for it.

HAIGH: No, I know of the Enigma machine.

KAHAN: Ah, the Enigma machine. It was a machine where you had to type each letter, one by one. As you typed each letter, a light would glow, and that would glow under a different letter, which you would now write down. So your plain text, as you typed it, would get turned into an enciphered text, which you would then send out in Morse code. And the guy who received it, he would receive the Morse code transcribed in paper. He would then type the thing, and the lights that lit up then would be the plain text. The Lorentz machine was a teletype machine. You would type your message on this teletype machine. It would do the encoding, and it would send the stuff out on the radio immediately. At the receiving end, you'd receive the signals, and you would use them to punch up a tape, he taught me geometry in my sophomore year; but Tutte was interested in graph theory, in things call matroids. And graph theory was recreational mathematics in the eyes of Beatty—in the eyes of most mathematicians at the time. I mean, you know, Euler's solution of the Konigsberg Bridge problem, well, it's about a river that runs through the city of Konigsberg, and there is an island in the river. There's a certain number of bridges and the question is, can you make a tour crossing every bridge just once? Euler showed that, yes, there was a solution to this problem. The name *Euler* is associated with many things in graph theory.

This was a recreational mathematics then, and so Beatty looked down on Tutte. We thought Tutte was a delightful character. I mean, he had a certain sly wit about him, so even though he was teaching us analytic geometry, the Cartesian geometry, he would stick in jokes. They were mathematical jokes. They weren't the sort of jokes you go "ha-ha-ha" about. They were the sort of jokes where you-- what can I say? You see mathematics in a certain way, and there are some parts of it that look funny. He managed to present some of those parts without it looking like a raucous joke. It was an "in joke" in each case, so he had a delightful mind. And we knew nothing about what he had done during the war. I only found out about it years later.

I don't know how Fritz Bauer found out about it. Fritz Bauer published a book in the 1980s on cryptology, and he has the mathematics of cryptology and the gossip about these things on facing pages. Well, his original was in German and then got translated into English.

I don't know how he found out all this stuff, but I did see Tutte. I got an honorary degree at Waterloo and, on that occasion, I did have a chance to chat with Tutte again before he died. He wasn't promoted because his letter, *T*, was too late in the alphabet. So Stanton and Tutte went to Waterloo and subsequently, when Davis sent them a lot of money, they built up a department. It wasn't just a department, it was a whole school of mathematics. It had optimization, statistics, actuarial science, and computer science all in one big school. They got a huge computer, the IBM 360 model 75. In Toronto, the best we could manage was a 360 model 65. They had computer time coming out of their ears. That's how they built the WATFOR compiler for Fortran and other things, because they had oodles of computer time, whereas we still had to stand in line.

I think the University of Waterloo now is preeminent in undergraduate mathematical instruction. They ace the Putnam exams quite often. I've been running the Putnam exams here at the university for about a decade, except for the past year when I was on sabbatical. So I've been watching, and the guys at Waterloo stand up there right up at the top, along with Harvard.

---

which you would then read into a standard tape reader for teletype machines. And it would all get decoded. It typed out the plain text. Of course, this went a lot faster.

Some German radio operator sending a message to somebody in Holland from France sent out a message, and the folks from Holland said, "Wait—this has gotten garbled somehow. Please send it again." And he did send it again, but he made a few changes. I think he put in some abbreviations or whatever, and somebody in Britain noticed that these two texts were almost identical and figured out what they must be about. And Tutte was told, "All right, here is the text as it was transmitted. Here is what we think the text says, at least in most parts. What the heck are they doing?" And he figured out how the machine must work from that, so they built a simulator for the Lorentz machine without having ever seen one. Now, all they needed was the key, and for that, they built Colossus. This was an electronic computer built to Turing's specifications, not entirely, but mostly, into which they would run attempts. They would take messages as they received them, and they would scan various keys until the statistics turned out right. And then they knew that they'd gotten the message. By the end of the war, they were deciphering these things faster than the Germans, and Tutte was the one who figured out how the Lorentz machine must do its encryption.

[Tape 2, Side A]

KAHAN: Okay, so when you asked about whether Toronto was preeminent, yes, it was then. And that's not saying a hell of a lot, but I think Waterloo offers a rather better undergraduate education in mathematics, and maybe even in computer science now, than Toronto does. The graduate situation may be different. I think Toronto is more prestigious still in the graduate school. Whether it deserves that prestige is hard for me to say, because I've just not been there for long enough to make a reasonable appraisal. Although Toronto was preeminent in Canada, its mathematics curriculum was old-fashioned. Chandler Davis is the guy who changed that. He was brought on board in I think 1954; maybe it was '55. That's when I was a graduate student. Chandler Davis moved much of the syllabus from the graduate courses into the undergraduate courses and modernized the syllabus very substantially. Should I say more about Chandler Davis?

HAIGH: Would that have been taking place after you returned?

KAHAN: No, this is while I was a graduate student, '54–'58.

HAIGH: Let's return to it in a little while, then, when we talk about your graduate school experiences.

KAHAN: Okay, well, ultimately Chandler and I were colleagues, and we had written a paper or two together, or three. I can't remember.

HAIGH: So it seems that, before we move on, one of the things we should talk about is the process by which you did become aware that there was such a thing as an electronic computer on campus, the FERUT.

KAHAN: Oh, yes.

HAIGH: I think you had mentioned briefly your tendency to take a part and recalibrate the physics equipment?

KAHAN: Yes.

HAIGH: Is that what led to your job maintaining it?

KAHAN: No.

HAIGH: No, so how--?

KAHAN: In my third year, I lost my scholarship. I did rather badly. I remember writing thermodynamics exam in which I might have spelled my name right, but if I did, that was the only thing right on it. The professor who taught that, she left-- I did really badly in my third year, as I said.

HAIGH: Were you distracted by other things?

KAHAN: Oh, yes. You saw her. I didn't appreciate that partial derivatives are part of an ambiguous notation. That is to say, what you mean by *partial derivative* depends much more on context than I had appreciated. And I persisted with that misunderstanding right through the whole year of thermodynamics. It baffled me, and finally, the year after that, it finally dawned on me! How could I have been so stupid? But, at the time, my acquaintance with other women, very casual acquaintance and observation, led me to the conclusion that this girl whom I've already described stood head and shoulders above the others; what's more, I wasn't the only one who thought so. Therefore, something had to be done to get an exclusive arrangement. But Sheila, who is very sensible, wasn't so sure that she wanted to be committed at that stage, and practically all my waking hours were devoted to trying to figure out ways to persuade her. Well, obviously I did, ultimately, but it definitely left me with no time to think about the things I should've been thinking about. It was a disastrous year.

HAIGH: Did you marry while you were still in undergraduate studies?

KAHAN: No. However, one strange thing happened: I was on the Putnam team that year, and I stood fairly high in the Putnams. I even got a little silver metal, somewhere in the top 20. I can't remember the exact number. My mother realized for the first time that I had some such ideas in mind when I didn't give her the medal. I gave it to Sheila, who made a pin; she still has it. So finally things worked out all right, then, and all things considered, I think I made the right decision; I am what I am today as much because of my wife as because of anything else. I owe her a lot, just support when I need support, moral support. We're well beyond mere companionship.

So there I was without a scholarship, and money was hard to come by. My parents weren't really rolling in it. So, I took on a part-time job fixing equipment for the physics department. I was fixing their electronic equipment, generally. Now, I would of course fix laboratory equipment that some of my contemporaries had broken. I'm sure they didn't break it intentionally in order to keep me solvent, but I could fix the moving coil meters. My eyesight was very good then, with suitable glasses. I could fix the delicate moving coil meters, which got bent around the pin, and I could straighten it out and rebalance it and stuff like that. I could fix the electronic equipment that physicists were using. Much of it was salvaged from World War II equipment of one kind or another, for generating high voltage pulses. And that would be done late some afternoons and often on Saturday afternoon.

That's how I got into that stuff, without the slightest inkling that a computer existed until the early spring of 1953. That was in my third year, and somebody said to me, "You know, there's a computer in the south wing of McClennan Laboratories—the physics building." And I asked, "What does a computer do?" And whoever it was, they casually told me. I wondered, how would you ever get a machine to do things like that? It has a memory, and you write instructions into the memory. And you write data into the memory, and it executes the instructions on the data. So, I decided I had to design a computer. I drew up flow charts and drafted diagrams of circuits for a relay computer. It had what we would now call a bus, which I construed as a kind of a railway line. And it had functional units, which I thought of as things like the roundhouse and sheds and so on and a railway line. It had switches that would switch information from one unit onto the bus to another unit, and so on and so forth. It had a memory made up of relays. After I'd drawn up the plans for this thing, I calculated how much power it would consume, and I realized that would be a substantial fraction, if not all of the output, of Niagara Falls. But, this is just a hypothetical device, and after I'd figured out how this thing would work, I walked into Kelly Gotlieb's. That's C. C. Gotlieb—Calvin C. Gotlieb, but he was called Kelly.

I walked into Kelly Gotlieb's office, cold, and I said, "Here's mine. Now, would you show me yours?" And he did. And that really changed my ideas of what I was going to do because it combined electronics and mathematics. That was altogether different from the stuff that I would subsequently be doing, repairing the physics department's equipment in my fourth year. None of that equipment had anything to do with computers, and I would have repaired that anyway, had I known absolutely zero about computers at that time. But in the spring of 1953, I learned that computers existed; in the summer of 1953, I partook in their summer program. I think that was the first year of the summer program. We were told at the outset that we would spend a third of our time learning to program and two-thirds of our time doing things ranging from sweeping the floor to punching tapes and so on. If we were good at programming, they would reverse the ratio. It didn't take long for them to reverse the ratio for me but, before they did that, I remember designing and building tape winders out of Meccano parts. They had lots of Meccano parts, because there was a differential analyzer, the so-called Vannevar Bush Differential Analyzer, an analog computer made of Meccano parts in one of the rooms on the second floor of the physics building.

HAIGH: And you'd also been unaware of this machine until then?

KAHAN: Absolutely. Actually, there was a similar machine that must have existed at Cambridge, because when I visited Cambridge later, in a basement I saw the remnants of the same kind of machine, the same torque amplifiers and everything. Fortunately, you see, as a child I'd had Meccano parts. My parents had gotten a Meccano set, so this was all familiar territory except for the torque amplifiers, which were really devious, very clever. But the rest was familiar territory. They weren't using the differential analyzer anymore. Why should they? So, they had a lot of Meccano parts left over, and we used Meccano parts to make tape winders because spools of tape went out. Everybody who was using the computer would carry around these spools of tape of various sizes, into which they would punch the programs of their data.

HAIGH: Right, and this would be five-channel paper tape?

KAHAN: Exactly, five-channel paper tape with a sixth hole for the sprocket…Creed teleprinter tape. Actually, those tape winders were still in use in early 1958 when the computer (which was called FERUT for FERranti University of Toronto), those tape winders were still in use. But I also learned to program, and I did some programming jobs. I think that was the year when I did some computations and tended to help General Electric design transformers. They had some formulas, which related the size of the iron core and the size of the copper windings and things like that to the energy losses. They also had some empirically derived formulas for the rate at which heat could be taken out. It was important to design the transformers with as little material as possible without getting it to overheat. The problem wasn't that they melted. The problem was that they'd char the insulation, which was just enamel.

So it was an interesting computation with a large number of formulas, and that's where I first became acquainted with floating point. The year before, a physicist who was interested in computing—that was J. N. P. Hume. Pat Hume had written a floating point interpreter for FERUT, and that meant that you could write a program that would do floating point computations—not quickly, but a hell of a lot faster than you could. That was what was being done. Each run was typically limited to five minutes. It was a schedule, and people's names went up on the board. And that was the order in which they would get their turn at the computer. When your turn came, the person ahead of you had cleared away their papers and tapes and stuff, and you'd slip your tape into the readers. Now, the machine was yours. You'd reboot it, and you run it from scratch. You get about five minutes, which was enough for, perhaps, 100,000 arithmetic operations. If it was floating point, it was rather fewer, maybe a couple thousand.

HAIGH: So there wasn't a separate staff of operators? You basically could sign up for a time slot on the machine yourself?

KAHAN: No, we had people who would punch the tapes for you, but when you were on the machine, you stood there naked in front of your God. The screen in front of you was lit with the little binary bright and dark spots, eight of them for the so-called B-lines, the sort of index registers, and two of them for two of the eight cathode ray tubes. They were repeaters. They weren't the ones that actually had the memory. They were repeaters that repeated what the others were doing.

HAIGH: So this was using the Williams tube memories?

KAHAN: Exactly. And that summer, the people who taught me programming, Beatrice Worsley, I think it was Cecily Popplewell, and, later, Joseph Kates.

Now Joseph Kates is an extremely interesting character. He had escaped from Czechoslovakia ahead of the Germans as a teenager. He'd escaped to England, where he was interned as a potential enemy alien. In the intern camp, he had the good fortune to be exposed to a number of others who taught classes, so he learned his mathematics and physics and so on at the internment camp. Then, before the war had ended, the British were shipping these guys out to Canada. He ended up in Canada, and I think he took some high school exams and was among the first admitted to the University of Toronto when the quota system against Jews was lifted. Now, the quota system was unspoken. It was just understood that if you admitted people according to their quality on exams alone, the place would be full of Jews. So they only allowed a certain limited number, but Sam Beatty—to his credit—realized the injustice of it all and said to hell with the quota. And that, I think, is how Joe Kates got into the university in…it must have been 1945 or thereabouts. Also, a lot of GIs came, but we didn't call them GIs. These were vets. You remember the GI Bill. We had an equivalent in Canada. So a lot of vets came into the university, and they were serious. They really changed the tone of the university. I don't think people appreciate how drastic the change was from before the Second World War to after. You see, before the Second World War, the university was not for scholarship. The university was where people of a certain class sent their children to meet other people of a similar class, or maybe they might try to work their way up. And they tolerated scholars. That was the excuse, but it was really a social function. When the vets came in, life was serious. These guys really did want to learn about things. It wasn't just a social occasion for them.

Joe Kates pursued an accelerated program, because he'd already taken the equivalent of a large number of the university courses. And he was working on his Ph.D. when he built (I think) the first operational computer in Canada in the basement of a mining building in 1948. In 1953, he took me down to show it to me, so I saw it there. It was still there. It wasn't really operating anymore, but it was still there. That was his Ph.D. thesis. It was improving what was called the read-around ratio. You've mentioned the Williams tube. You know how the Williams tube worked—by secondary emission from electrons. So if you accelerate electrons hard, they splash into the face, and they splash out more electrons that are coming in. You get a net positive charge. If they come in slowly, they fill up these potential wells, and then you get a net negative charge, and that's how you stored your data. And how did you sense the data? Well, you would send in a somewhat feeble stream of electrons. You may notice that there was a change capacitively coupled. You may notice there was a change in the voltage because there had been a well there of positive charge, and now it had gone negative. Then, that was a one. And if you didn't notice any such change, that was a zero. Okay, sounds great. But, you see, when you did these things, you'd splash electrons into adjacent cells, so there was something called the read-around ratio. How many times can you read things without splashing electrons off into adjacent cells to fill them up improperly? You would have to redo the stuff, very much like DRAM today. See, DRAM has decaying memory. You've got to read from the DRAM and rewrite it back frequently enough, or else you lose it. And that's what the cathode ray tube is like. And Joe's thesis was about improving the read-around ratio. He got it up pretty high. He got a very reliable little machine. But the people who were running Canada at the time really didn't want to have a machine designed by some Czech immigrant. They wanted a machine designed by reputable engineers, so they bought their first computer from Ferranti, using a design from Manchester University.

HAIGH: So this small machine in the basement actually ran programs?

KAHAN: Yes, little programs, toy programs. I think he only had an 8-bit wide sort of thing. It was a very tiny little machine. It was just to demonstrate the feasibility of the concept of getting a high read-around-ratio from your memory. It was a hell of a lot more reliable than FERUT ever was. But maybe this is a good time to break, and we can go and have lunch.

Session Two begins on the afternoon of August 5, 2005.

KAHAN: Okay, we were talking about Joe Kates?

HAIGH: Yes. We were. I believe you described this small machine built in the basement.

KAHAN: Yes, of a mining building. It was across the street from the physics building at that time.

HAIGH: And I think this had been in the context of your programming course that we had come from, and then you had mentioned that Kates was remarkable and began to talk about the computer?

KAHAN: Yes. Okay, so initially I learned programming from Trixie, Beatrice Worsley. Her nickname was Trixie. And I learned from Cecily Popplewell, although I had very little contact with her. And later, when it became pretty clear that I was good at this stuff, I came under the wing of Joe Kates. Joe Kates was a prodigy—a phenomenal programmer; incredibly devious. I'm going to give you an example. Because ours was a binary machine, there was a good deal of time spent on binary decimal conversion both ways—decimal into binary and then binary out to decimal. Joe wrote our best binary decimal conversion program. Actually, it would handle any of several radices. You could use a radix different from ten or two, within limits. But the important question was, are you going to decimal or from decimal?

Now, what you have to know about the Ferranti Manchester Mark One is something about its instruction set. It was very peculiar. Each instruction-- now, I hope my memory has served me correctly. That's why on that sheet, you're going to see "stroke-E-+-A-:-SIU-half-DRJ." That's the Creed teletype code. It had 32 characters, and I can't remember exactly what the last eight of them were, so that tells me that my memory is not 100 percent. I may not have everything exactly right, but I'm going to try. The instruction word was 20 bits wide, although the address went to each instruction word, in fact, the typical data word was 40 bits wide. The eight index registers had a 20-bit wide integer in them. Every instruction (without exception, if I remember rightly) was indexed, so that every instruction had a three-bit field, which designated the index register whose contents would be added to the instruction before it was executed. Now, when you add 20 bits to 20 bits, needless to say, some of this slops over into the op code. Yes, you could put something in an index register, and it would alter the op code, not just the address. And Joe used this. If you put a certain constant into index register zero (which is what was being used by instructions that initially, when you look at them, worked with indexing) then your conversion was from something to binary. And if you put something else in index register zero, it was from binary to something else. Needless to say, this was tricky. I mean, this was beyond tricky, and I can remember having spread the code and a flow chart out on the floor to see if I could figure out how the hell this thing worked. In its own way, it was an education.

So Joe was a virtuoso programmer, and I think that that set me on a certain path to history. No, what was the standard for ingenuity in programming, just the act of programming as distinct from figuring out what the program was supposed to do, you see--

HAIGH: Was the coding being done with the aid of an assembler, or was it straight into--?

KAHAN: No, not at that time. There was no assembly. We were coding in machine language directly, in the sense that each Creed character gave us five bits. So "t-stroke" was "load the accumulator," I think. "T@" was to load the accumulator from a funny check bit line, and so on. You just had to memorize these two character signals. These were the op codes. But in each of them, three of the bits designated the index register. Therefore, in the 20 bits, you had ten bits for the op code and index register, and ten bits for address, except that we had only use of eight of those bits. We only had 256 words of cathode-ray memory. We could've gone up to a thousand, presumably. At least, the architecture allowed for that, but we didn't have that much memory around. I don't think anyone ever did. I don't know. So, it was Joe who taught me deviousness. That was a very fruitful summer, and when I then entered into my senior year--

HAIGH: Let me ask, in the programming course itself, do you remember what kinds of things you were being taught?

KAHAN: Well, we could look at the boot sequence to learn something. The way you started the machine was to clear everything. When you cleared memory to zeros, the zero on the memory was "read the keys." You set into the keys whatever you wanted to do in order to begin the boot sequence. That might read a little loop of tape. That little loop of tape, once read in, would now give you the initial instructions that would then start the whole thing going. There was also a subroutine-calling convention that we had to learn—JSP I think it was—which meant "jump to a subprogram," recording the return address in a suitable way. But this required that one of the pages in memory had a table of constants, the powers of two used for shifting, because it was a serial machine. So you shifted by multiplying. Yes, that was remarkable. This machine had a one-millisecond cycle time for add, subtract—the usual instructions. But I think it did a multiply in three milliseconds. It had a lot of vacuum tubes. When multiply would go out, I remember all those vacuum tubes had things to do with them.

So, we learned some of the fundamental stuff, which perhaps wouldn't have been the best way to start a programming course, but that's how that one worked. Then, we'd learn how to do various simple tasks, like read in a bunch of numbers from a tape, add them up, or combine them with something else. It was very simple things. And then, almost as if you were being taught to swim by being pushed off the dock, we'd get some real things to do from people who had real problems. We were supposed to know about programming because we'd memorized the instruction set, you see, or more or less memorized it. But that was the difference between us, who knew something about what the machine did, and a supplicant who came with the problem who had no idea what the computer did. In fact, many of them had no idea what an algorithm was. I think I understood instinctively what an algorithm was and, at first, I was baffled why other people didn't understand instinctively what an algorithm was.

What makes it an algorithm is distilled wishful thinking. You know what? There are still a lot of people like that who don't know the difference between an actual procedure that you can follow and wishful thinking. They think you just have to say some magic words, and anything you want can be done. Computers aren't quite that way. They do what you tell them, not what you want. Anyway, I don't think it was very long until I was writing programs that produced results that people wanted, and other people didn't fare quite so well. I mean, there were some folks who seemed unable to get the hang of it, so they spent more time punching tapes. You remember that ratio one to two versus two to one.

HAIGH: Now, you mentioned how impressed you were by Kates's ingenuity in programming; as you were writing your own programs. What was your personal, emotional kind of reaction to the experience of programming?

KAHAN: I was so relieved that the machine was doing it instead of me. That is, I had learned to use various electromechanical calculators. I'd learned well enough that, while I was a graduate student, I actually held custody of several of them—one of them at home. And when I became a professor in 1960, I was in charge of the math department's collection of electromechanical calculators. It was so much so that one day a salesman came by to sell us the newest and latest electromechanical calculators, and he was selling good ones. They were well-built calculators. I said, "I'm not buying anymore, not until you have an electronic machine that costs less than a car." And the salesman was outraged, and he went to the chairman of the math department to protest that "he was an unreasonable guy. He wouldn't buy any of the calculators." And the chairman at that time was perfectly willing to leave that to me. But yes, I knew how to use electromechanical calculators.

My room as an advisor, which I started doing that summer and persisted with off and on until I got my Ph.D., was initially the same room as had the 407 programmable card punch. I learned to program that with the plug board, which made a hell of a lot of noise. Then, there was a sorter that made a lot of noise, and people would come in with their problems. They'd want advice and, every now and then, someone would come in with a deck. So I'd have to deal with somebody who wanted advice over the din of these machines. But it just seemed natural. I don't know how else to put it other than it's as if I were born for this. I understood programming; I understood the various machines; I knew how to fix them; I could fix a card jam; I could fix the Creed teleprinter. Sometimes—not always.

We had a tape reader. This tape reader deserves mention. The tape reader, at full speed, would read one character, a five-bit Creed character, every five milliseconds. It could go at 200 characters per second. A tape whipping through there at 200 characters per second, you'd think it was a lethal weapon. But it hardly ever went that fast, because the time required to assimilate the character on a machine whose cycle time was one millisecond meant you had only five instructions in which to accept this character and deal with it before requesting the next. If you delayed in requesting the next, then a magnetic clutch would stop it. A magnetic clutch is essentially a chamber, filled with oil and magnetic powder, and it has two spinning disks. One shaft is driven, and the other is driving from the motor. And when you want to connect the two, you impose a magnetic field, and that magnetic field lines up all these little iron particles. So, it's as if you have a solid connection, and when the magnetic field is off, these things just dissolve immediately. You've got to choose the right type of material so it doesn't become permanently magnetized. So, they could start and stop this thing within the five-millisecond window. But then the tape would stutter, and it would take a while to start up again because there's a certain inertia.

So if you couldn't absorb the character in five milliseconds and then be ready for the next, the tape would run a great deal slower. I think I was the only person who ever wrote a program that actually ran the tape reader at full speed. That was only for reading in integers. I think they were six-digit integers, spaced eight characters apart. That is to say, each six-digit integer took eight characters, something like that. That was later. I needed it for my thesis because I had a lot of numerical data to put in in order to initialize things. So I wrote the program for that purpose, and it went so fast that, if there was the slightest flaw in the tape, even including a fold, the tape would break. So I had to make sure that my tapes were flawless, and I had to make sure they went into a bin where nothing bad would happen to them so I could rewind them later.

Well, yes, so I was mechanically adept. I think I understood pretty much everything that was going on and around the computer. Later, the engineers were willing to let me take the computer—this was when I was working on my thesis—I could take the computer over on Friday evening, and relinquish it on Monday morning; they would allow me to bring the computer down to replace a vacuum tube if that's what it took. When the computer was cranky or just not working I could often recognize which vacuum tube was causing the trouble. We would have a Cossor oscilloscope or some diagnostics programs to run. I don't remember all the details, but I do remember that there were one or two occasions when I brought the machine down, replaced a vacuum tube, brought it back up again, and continued. People think it's just a switch. Vacuum tubes have two supplies: one to warm the filaments called the *A voltage*, or *A supply*; and one to supply the anode voltage, which is typically a much higher voltage. But, if you just turn things on and off, the shock can actually shorten the lives of the vacuum tubes. So the first step was to bring the B voltage down gradually; after that, to bring the A voltage down gradually; and then go and replace whatever had to be replaced, and then reverse the process to bring it back up again. The engineers trusted me to do that. I had watched them, and they were satisfied that I wouldn't break their machine.

The disk was another matter, 30,000 words on the disk of which maybe 10,000 or 15,000 were working on any particular day. They had to make up a directory of which tracks were working that day, and the directory would be posted. You'd have to punch up a little bit of tape to do the translation, so that the tracks you were using in your program were virtual tracks and had to be translated to actual tracks. It's a little table, you see, because some of the tracks weren't working. And that would change from day to day—interesting? So you can get the idea from this that somehow I really was in my element.

HAIGH: So it sounds then that you weren't someone who liked programming just because of the clean, logical abstractness but, rather, because of this interaction between things that you could control through the code and the various pieces of machinery themselves. You liked the physicality of the machine.

KAHAN: Well, it was all of it. I liked the mathematics. Of course, I liked elegant programs. I would like short programs that are just as fast as the long ones or faster. The elegance that I learned to appreciate from Coxeter played a role every day; every program revised and so on until it was a gem in its own right; an investment out of all proportion to the transcendental value of the code. It was just because I liked to see something crafted well. I had a feeling for the machine from the ground up.

Let me give you another example. This machine used a large number of vacuum tubes. Where would you get a large number of vacuum tubes? Well, there were warehouses full of war-surplus vacuum tubes that had been intended for use in radar sets and aircraft. Somebody had gotten the idea that glass envelopes for these vacuum tubes would add to the weight, and that they could reduce the weight by using aluminum envelopes for vacuum tubes instead of glass—lighter vacuum tubes. This posed certain problems for the sealing, so you have to put seals around the electrodes, which had to go through little glass beads, and finding the right kind of glass and intermediate stuff. So it was a technical problem of some interest. And we had warehouses full of these vacuum tubes in England, and these are the tubes that Ferranti decided to use on the computer because there were so many of them. Well, unfortunately, aluminum is permeable to hydrogen—something that the people who advocate hydrogen-fueled vehicles seem not to appreciate. Gradually, these vacuum tubes would get gassy. Instead of having a vacuum, you'd have hydrogen in there; hydrogen can be ionized at voltages as low as 12 volts. So inside the vacuum tube, if only you could have seen it, it was glowing not only red from the filament, but purplish-blue from the hydrogen. Of course, such a vacuum tube, then, was no longer acting as the switch it was supposed to be, and it would get terribly hot. And the heat would merely accelerate the process of percolation of hydrogen. You know, there isn't a hell of a lot of hydrogen in the atmosphere—but such as it was, it would get in there.

The guys who had designed the circuits had used what are called *cathode followers*. What *cathode followers* means is that the output from the vacuum tube's anode is a voltage that fluctuates through several hundred volts as you change the logical state of the tube. But it does so across an impedance of several hundred thousand ohms, which means that you can't load the output heavily without damping it out and killing it. So, that means the output from one vacuum tube can't be expected to drive lots of others.

[Tape 2, Side B]

KAHAN: The output from the anode is at too high an impedance, and you can't load it with many other vacuum tubes' inputs. So you have to use a cathode follower, as it's called. And what that does is it mimics the voltage transition at the anode, but it mimics it at the cathode of the cathode follower, where the output impedance is lower and you can supply more current. Therefore, you can drive more wires and more inputs to other vacuum tubes. But the guys who designed this used very high resistance loads in the cathode circuits, and they did this in order to reduce the amount of power that the computer was consuming, and to reduce the amount of heat that they would have to get out. We had air conditioning for the cabinets—just the cabinets—of the computer. It was cold air circulated through them, and that kept the cabinets somewhat cooler. So actually, the computer room was cooler than the other rooms during the summer. But if they had run these things at a lower impedance, which meant that you would have had higher currents going through the vacuum tubes, that would have generated much more heat and consumed more power. It consumed enough as it was. I'm trying to remember how much. I think it was something like 400 kilowatts. That's a lot of power: 400,000 watts they were running.

That power was supplied by a motor generator unit. The fact that it's a motor generator will figure later in our story, but it meant there was a motor driven off the electric means supplied by Ontario Hydro, most of it from Niagara Falls. That motor spun a flywheel and a generator, and the generator generated power at 400 hertz. The input to the motor started at 25 hertz—60 hertz in most of the rest of the country, but at that time Ontario had 25 hertz power. Well, that was one of Lord Kelvin's mistakes. It's an interesting story, too, for perhaps another day. And the 400 hertz power meant that you could distribute the power at various voltages with smaller transformers. Twenty-five hertz takes a lot of iron in the transformer. Four hundred hertz takes much less, and 400 hertz was often used in military equipment. So there were a lot of these transformers around.

Well, unfortunately, because of the high impedance of the cathode followers, they were terribly vulnerable to gassy tubes. So they would often malfunction and, furthermore, they were vulnerable to all kinds of interference. So later I'll refer to the fact that the mean free time between the mistakes made by the computer was about five minutes on a good day. At five p.m.—when industry turned off and housewives turned on—you frequently couldn't run the machine for more than 30 seconds without it crashing. Why that happened is a mystery to this day because, you see, the electric power lines went to the motor. And then there's a big, fat flywheel with a rubber coupling between that and the 400 hertz generator. How were the transients coming in off the power line and getting into the 400 hertz supply, and then into the rest of the computer? We never found out. We just never found out why that happened. I still don't know to this day, but I'll have a better story to tell you about it later.

So, this made the computer unreliable. Because of the very high impedance of the cathode followers, they did not absorb transient energy. They couldn't absorb transient energy and, consequently, you might as well not have had them. This is sad. It was a design decision made by Ferranti, and it so reduced the reliability of the machine as to destroy its commercial prospects. Univac took the market that Ferranti could have had, because Univac was much more conservative in the circuit design. They used a lot more redundancy and other things so that their machine was reliable—more than a magnitude more reliable. I learned something from that. You know, it's not what a machine or anything else does; it's what it does *reliably*. What can you depend on? That's what matters. But I'll give an exception to that later when we talk about the CDC 6600.

HAIGH: Do you think it would make sense to talk now about this project for Adalia Ltd. on the reservation simulation?

KAHAN: Right. Okay, so I'd gotten my bachelor's degree, and I see my plan to get married in September. And between then, starting in May, Joe Kates, who now knew that I was a fairly good programmer, hired me to simulate this reservation system. Now, the history of this is a little bit significant here. Lyman Richardson was a low-level executive and engineer working for Trans-Canada Airlines, as it was then; Air Canada, now. He was working in the communications area. An enormous amount of their costs went into communications, and here's why: if you wanted to book a flight, you would phone somebody. That somebody would take your request and try to see if it could be fulfilled. To do this, they had to make, typically, an inquiry of a central data bank, which was actually in a building in Toronto, where they had blackboards up on the wall for today's flights and maybe tomorrow's. Other flight information, seats available and so on, was kept in an enormous sort of a Rolodex file. And if you wanted to find out about flights, say, next week, there you are at the telephone. You've got a customer at the other end of the phone line. They've asked for flights, and you've looked up something to see which flight numbers might suit them. You now scribble down a request on a piece of paper. You want to know if so many seats are available on such and such a flight, such and such a date, and so on. You fold this up, and you stick it into a conveyor system. This conveyor system has a number of channels. This little piece of paper slides around on this conveyor system in this room until it gets to the guys who are in charge of the Rolodex. They take the piece of paper, flip through the Rolodex in order to see if there's something available, and then they're mark down whether it's available or not. They stick it in there, and it goes all the way around and back in this little conveyor system to the telephone operator, who knows that there's a certain color of piece of paper on a certain one of the channels that's hers. She pulls it out, and then she can tell the person at the other end of the phone line, "Yes, there's a flight available. Would you like to book it?"

Now, this is a horrible system, so if somebody phones up a local office in Vancouver of Winnipeg or Halifax, what they'd be told was, "Let me get back to you." And then, somebody there would consolidate a certain number of these messages and make a long-distance call or send a telegram to Toronto, where it's going to go through this process. Well, sometimes, it actually took a while for the guys with the Rolodex to find out, because they were getting these messages in clumps. They couldn't tend to each one immediately. If the messages were coming in clumps, they would pull a message out of one of the channels. They'd have to record which channel it was in to remember which channel to put it back in, and they'd get backlogged. So the telephone answerer would have to say, "I'll get back to you." Well, what if the telephone answerer has to say 'I'll get back to you' and it is, "No, we don't have any seats available"? Sometimes the seats were allocated in small blocks locally so that people could respond quickly. After they exhausted their allocation, they'd have to go through this process. Or sometimes there'd be a shortage of seats to satisfy somebody over here. They'd have to see whether some allocation over there could be used for the purpose. So it was communications, communications all the time, and it was terribly costly.

And the irony of it all was that the costs were with another Crown corporation. Trans-Canada Airlines was a Crown corporation. That meant that it was operating on the behalf of-- well, let's see, in 1954, that would have been Her Majesty the Queen. Queen Elizabeth came to the throne, I think, in 1953. Is that right?

HAIGH: Oh, well she had a jubilee just recently. That would fit. About two years ago they had a big party, and I think Paul McCartney sang something.

KAHAN: All right, so I think that was it. It was Her Majesty, so it was operated in Her Majesty's name. Canadian National Railway is another Crown corporation, also operated the telegraph lines. So it was taking money from one government department and putting it in the pocket of another but, nonetheless, it was regarded as an expense for Trans-Canada Airlines. And it was a whopping big expense. And Lionel Richardson had the idea that if they could just computerize this reservation system, then there'd be instantaneous response, and you wouldn't have all this communication going on. And my job trickling down, trickling down through Sir Robert A. Watson Watt-- It's worth reading the biography of Sir Robert A. Watson Watt because, you see, he came to Canada because his second wife had to come to Canada to take care of some ailing relatives. So he followed her to Canada, giving up his business in Britain, not giving up completely, but instead of supervising it setting up in Toronto, Montreal, and New York. She had to be in Toronto; he had to be in Toronto. She'd be passing through from time to time in the office, and he had a couple of henchmen. And they were the ones who got the contract but, of course, Sir Robert knew nothing about computers, nor did his henchmen. So they had to find a guy who knew about computers. That's how Joe Kates got into it, and Joe Kates needed a programmer. That was me. That's how it started.

HAIGH: So what you were doing wasn't intended as a prototype for a possible computerized system?

KAHAN: It was intended as a demonstration to see whether a computer could manage their reservation system in such a fashion as it would cut substantially the cost of communications. That was the whole business as far as Lionel Richardson and Sir Robert and Joe Kates were concerned. As far as I was concerned, it was a question of finding out what the hell they actually did. So I went down to, it was the Canada Tire building, I went down and looked over their shoulders to watch carefully to see how they operated. They had a number of special codes. First of all, there were the codes for the airports which, at that time, were two-letter codes. They're three-letter codes, now. And I knew what they all were.

HAIGH: All the Canadian airports have *Y*s.

KAHAN: They start with a *Y*; that's right. That distinguishes them from the American ones. Vancouver used to be *VR*; it turned into *YVR*. Toronto used to be *YZ*. It turned into *YYZ*. I'm sure you feel comfortable about hearing *Z* being pronounced as "zed" instead of 'zee,' but, you know, that's how we Canadians do it.

So I knew the route used looked a bit like the skeleton of a fish. Most of the routes ran east to west. There were spikes going up the north, and a few spikes going down to some American cities. They had landing privileges in New York, in Chicago and, I think, Seattle at that time. I knew the routes. I got to understand the schedule well. I understood the abbreviations for the teletype transmission, whether they have reservations or not. I understood the limited ability to suggest alternate routings. I understood the limitations on what are called long-haul flights and short-haul closure. That meant that if there was a flight that went from Halifax to Montreal to Toronto to Winnipeg to Edmonton to Vancouver, that was not the sort of flight in which you wanted to sell somebody a seat going simply from Toronto to Winnipeg because it might foreclose a much more profitable sale. But when the time came that somebody wanted such a seat, and it didn't look as if anyone was going to take long distance, then you could. So there was something called short-haul closure, initially, on these flights. I came to understand the business practices, and that took almost a month.

Then, I was writing flow charts about what the procedures would have to be. Now, remember, I was dealing with a machine that had five minutes between mistakes on average, and I knew that for a demonstration, which was scheduled for the end of August, a demonstration in which the machine crashes is the last thing you want to have happen. So, I had to put in an enormous amount of what would now be called checkpointing, redundancy check sums. Every transaction was recorded. A simulation of the transaction was done in order to see that the transaction could actually take place. Then records were updated permanently. An attempt was made to ensure that the period during which an error on the computer, if it occurred, would really screw us up. But that was minimized, by doing various things redundantly and checking. I mean, checking, repeating, checking redundancy all through this horrible thing. It was done so that, maybe, there was one instruction in a thousand or so where a machine failure would really louse up the system. Otherwise, I could recover. There was a little loop of tape left in the tape reader, so that if the machine did crash, it would normally go to somewhere where there'd be a beeping sound to tell me that something had happened. Then the tape would go *zoop,* and the thing would reboot. It would find everything's on the disk, and it's all in order, then look to see whether the transaction had been consummated or not. I remember doing the demonstration, which occurred later. Somebody asked, "Why did the machine just beep there?" And I said, "Well, the machine caught a mistake and fixed it," which left them rather impressed.

But that was the big problem! It was to make the machine look reliable when it wasn't. So I had these very elaborate flow charts, and it was getting to be early in July. I think it was somewhere near Dominion Day, which is July 1 in Canada. Independence Day is July 4 in the U.S. Somewhere in there, Joe started to have kittens, because all he saw was flow charts. Where the hell was the code? I said, "Don't worry about the code. The first thing to do is to figure out what we must do. The code will write itself."

But he lost confidence, so he brought in a physicist friend of his, Joe Shapiro. And Joe asked me, "What's going on?" and I said, "Well look, I've got all these flow charts, and I think we're pretty much finished with flow charts now. So if you'd like, you can help me translate them into machine code." And he did, and it only took about two weeks or three. And then we had the program. And not only that, the program was working because we could understand what everything was supposed to do. The flow charts were broken down in such a way that if something happened, you'd get an announcement of some sort that would tell you where it had happened. So you knew where to look.

We found ourselves with time on our hands, so the first thing we did was to make up some gorgeous manuals. It looked exactly like the manuals they were distributing to their telephone answerers. Then we said, "Well, gosh, we still have time, so what about management reports?" If, for example, more flights, more seats are canceled than had ever been booked, that's something that we should notify someone about. Or, what happens if there's a sudden run on seats? Seats are being booked at an unusually high rate. Shouldn't they put on an extra section? That is, they'd fly two aircrafts with the same flight number to accommodate extra passengers. Shouldn't there be something that enables management to add a flight if they want or delete a flight if they want and so on? So we looked at their operations manual. We duplicated damn near the whole thing, and it was in the language that they were using at the time. So we could take one of their people and put them at the console, at the Creed teleprinter and so on, and they'd think that they were just communicating by telegraph to somebody, doing usual stuff.

Finally it was time for the demonstration. It was late August, and I had a somewhat red sports jacket made out of a Scottish cloth. It was not all red like that. It was red with flecks of green and brown and yellow in it, but if you call it a red sports jacket, you are not too far off. Joe had a green sports jacket of somewhat the same flavor. I mention this only so you understand we were not wearing gray suits.

HAIGH: Yes, you see the pictures of the systems analysts from that era. They always have dark suits, kind of thick, black glasses, and kind of skinny dark ties, I think.

KAHAN: Joe and I were not like that. Our turn to come on stage appeared. There was this introduction by Sir Robert, and then Joe Kates had a couple of words to say. His English had a lisp in it. Then it was for Joe and me, and we explained what they were going to see. And what we said was essentially this: what you're going to see is a reservation system exactly like yours, insofar as you can tell, with one difference. Every request will be handled within two seconds. That's the only difference. The computer makes that possible. All right, so these executives come. They crowd around the machine, and they watch while one of their folks enters things. Enter reservation for a seat on flight so and so. Someone says, "Enter a reservation for a seat on flight such and such," knowing there is no such flight. Back comes the little message of, "No such flight," or "Let me reserve so many seats on this flight," and it turns out to be more than its capacity, and he's told that. We had even gotten to the point where, for some requests, we could figure out alternate routes. Remember, the route was a very simple route, so there were some cases where, if there were no seats where the guy wanted to go, and we knew there was a way of going. Say, instead of going from Montreal to Toronto, you can go from Montreal to Ottawa to Toronto. Now, that's more expensive, but if you really want to go, you can go that way. So we had a few at a table, and yes, that started to come out. "What happens if I want to cancel a flight?" Sure.

We thought that this demonstration was a smashing success. It did everything and more, and it did it in two seconds. And every now and then there'd be a "beep-beep," and the machine would correct itself. There'd be a beep, and then the loop of tape would go, and then you'd be back up. We were so wrong. Oh, when I think back on it, we were so utterly wrong. These guys had devoted their lives to building up the system as it was, and they saw it being snatched away from them by these two guys in garish sports jackets! So, they'd have none of it. The contract was successful, and everybody got paid, but they didn't adopt it. They had to wait until these guys retired, some with a certain amount of encouragement and, six years later, when I had returned to Toronto as a young professor, Trans-Canada Airlines had a contract with Packard-Bell, I think it was, to build a reservation system. They came from my flow charts, which I gave them gladly. It's sad, in a way, because they'd have some historical interest for you. But I was glad to give them the flow charts. I really wanted to do other things, and they never did build a system. I think Trans-Canada Airlines ended up using American Airlines' SABRE system for a while.

HAIGH: So I have some follow-up questions on that aspect of that. So the first one would be, at that time, the Magnetic Reservisor system by Teleregister. It'd been operational since 1952 using some magnetic drums, I think, to store basic information on flight availability. So as you worked on this project, were you aware of the existence of that system?

KAHAN: No, we were aware that American Airlines was using something, but we weren't aware of what it was. And in any event, remember, this was a contract that had evolved over some period of time. I daresay that, when Lionel Richardson first put forward the idea of doing this, it took some years before it could get underway. It may be that, in the interim, other systems came up. He couldn't have been the only one in the world to notice that they were being killed by communications costs.

HAIGH: You also mentioned flow charting. I was wondering, was that a technique that you were taught in your introductory programming course?

KAHAN: Taught would be too strong a word. It seemed like the obvious thing to do, and I know that at IBM they had little templates for drawing flow charts. We never saw them until we got the 650, which was 1958. I don't know that anybody had to teach us to use flow charts. It just seemed to be the natural language. And you see, there wasn't the word *checkpointing*. There were check sums, but checkpointing, where you store stuff away, check to make sure that it has been stored away. Then you do something and, if it doesn't work, you can go back and so on. Nobody had the word *checkpointing*. We just did it. There are all kinds of things that we did without knowing that we might be pioneering, for all we knew, practices to become standard later.

HAIGH: So, it seems that you had enjoyed this process of investigating the business systems and discovering the requirements. Did you ever consider becoming a consultant, basically a management consultant specializing in computer applications, at that point?

KAHAN: No, I thought of myself then as I think of myself now: as preponderantly a scientist and, of course, when I got back to Canada in 1960, an educator, a professor. And if I had not chosen to teach, I might have considered what you described. It wouldn't have been alien but, you see, the Canadian government, which means the Canadian people, had been paying my way for a while. I had enjoyed a scholarship for three years. As a graduate student, I was paid to teach. I taught courses in the math department. Then I got this post-doc again from the Canadian government, the Canadian taxpayer, to pay my way for two years in Britain—some £1,500 a year, no taxes. That amounted to a lot of money in England. I felt that I should do something to pay it back first. I felt that I had incurred a debt and that I could easily get jobs. General Electric had wanted to hire me. I'm sure I could have applied to other universities, but I felt very strongly that I had a debt to repay to Canada and to Toronto, so I chose to go back to Toronto to teach. It wasn't just the path of least resistance. I mean, there was a place for me there, of course. It seems that I was appreciated. No, I owed them something. I didn't feel as if I made myself. I owed a lot to Kelly Gotlieb, to Byron Griffith, to Daniel Delury. There are all sorts of people who had, I felt, gone rather out of their way to make it possible for me to flourish. I just felt a sense of obligation, so I became a professor. I sort of fell into it.

HAIGH: We should rewind slightly, then. This had been something that you did in the summer.

KAHAN: In the summer of '54. That's right.

HAIGH: Between your graduate and undergraduate degrees?

KAHAN: And then got married. Now, why did I return to graduate work? Well, I think it's because Kelly Gotlieb recommended that I do that. There were jobs available, lots of them at that time. There weren't all that many people who knew what a computer was, much less how to work it, you see. So, of course, I could get a job almost anywhere. But if I had known what was going to happen in 1957, I might have chosen differently. But at that time, it seemed that I could be a graduate student, and I'd be contributing to the technological development of my country. It wasn't as if it was any skin off my nose doing that. It looked to me as if it were a win-win situation all around.

While I had been working at the summer course, and then some time after it, during fourth year, I was approached by Coxeter. In my fourth year, our senior year, Coxeter was teaching a course on non-Euclidean geometry, a standard course in geometry. I took it, even though it satisfied no prerequisites, as one would say here. It was simply because Coxeter was teaching it…and what can I say? It was beautiful. But we knew that Coxeter was interested in combinatorial geometry so, after a while, we got together as a class. It was a small class, with fewer than half a dozen students, and we asked Coxeter if he could finish up the non-Euclidean geometry and move on to the combinatorial geometry that we knew was closer to his heart—and which we wanted to find something about. Coxeter was just delighted. He was so happy that we wanted that.

And in the course of dealing with that, something came up about groups. Now, there are lots of ways to define groups. One way to define groups is by means of what are called generating relations. What you do is you write down a set of identities. It looks just like polynomial equations, and then you ask yourself, Is there a group whose elements satisfy these equations? And if so, how big is it? Well, how big is the smallest group, so to speak? And there's an algorithm for doing it. It's called the Todd-Coxeter co-set enumeration scheme. "Todd" is John Todd, who I think ended up at Caltech. I think they developed it while they were both at Cambridge in England, and I programmed that up for Coxeter. And he was delighted. I felt that I was repaying part of the debt, you see. The trouble was that the machine has only 256 words in storage. Doing it in the obvious way wouldn't handle any really large groups. So, although we could do groups that were bigger than you would have done by hand, it took only a moment to write down some generated relations for a group that was just far beyond the capacity of the machine.

So, when I graduated and thought what should I do for graduate work, I thought working on this problem would be an interesting thing to do. And I'd learn a lot more about group theory. But Peter Bandler wanted to work on it, too, for his master's degree. So we flipped a coin, and I lost. So he was going to try to write an improved version of the Todd-Coxeter algorithm, and I ended up working on the Laplace difference equation for solving heat flow problems and electrostatic potential electrons, and Kelly Gotlieb was going to supervise that. And it was timely, because there was a lot of work being done on that kind of thing at the time. So I wrote a master's thesis on that. I then discovered some interesting things, which distracted me. I didn't actually finish the master's thesis until 1956. In 1955, I'd already discovered some really interesting things, and the reason for writing the master's thesis was so I'd have something in case other things didn't pan out. Also, I was getting a little beyond what Gotlieb wanted to pursue.

HAIGH: Now, can you remind me, was Gotlieb in the mathematics department?

KAHAN: No, he was in the physics department.

HAIGH: So you were studying in the mathematics department, but with an advisor in the physics department?

KAHAN: Yes, that was okay.

HAIGH: Did anybody in the mathematics department have much to do with the computer?

KAHAN: Well, the statisticians did a bit. Jimmy Chung was a statistician. At that time, statistics was a part of math department. They broke away later, but at that time statistics was in the math department. Jimmy Chung was adept at computing. The actuaries, Bailey and Sheppard, were interested but incompetent at it, so they would like to hire someone to do their actuarial stuff. They didn't do programming themselves, you know, hard to teach an old dog new tricks, perhaps. The people using the computer were mainly physicists and chemists. It seemed to come naturally to them. It did not come naturally to mathematicians and, at first, it didn't come naturally to the engineers, either.

When I became a graduate student, I remember taking a course in 1956 on numerical analysis from Gotlieb—awful course. It wasn't his métier. I think he used Hildebrand, which came out in 1956. That's the text by Hildebrand. It was considered a modern text, but it wasn't. It was very much text in the old, conventional style. I also took a course on integral equations from A. J. Coleman. It was pretty pedestrian course. Taking that course, were a number of other graduate students, including some physicists and physical chemists, and they had some really interesting problems. I was the advisor-type still-- I think my room had moved. I was no longer competing with the cardpunch. I had a little closet up in the cloisters, a room about the size of a large coat closet with a desk and a little window. So people would come there for advice. My graduate student friends would come for advice, and I would work out things to help them. And all the while, I was working on my own thesis problem.

Now, what had happened was that while working on my master's thesis, I had discovered, as everyone would, work by David Young, David M. Young. He had written a marvelous thesis at Harvard under Garrett Birkhoff. And in his thesis, he showed that it was possible to accelerate enormously the convergence of what was called then successive over relaxation. I think it's worth explaining it a little bit because the ideas seem so simple, but the analysis turned out to be enormously deeper than we would've expected. Relaxation methods were invented by the guy by the name of Southwell, for dealing with the deflections of elastic structures under load. He wasn't the only one who did it, but he was the one who gave it the name of *relaxation*, if memory serves. Others did it, for example, A. C. Aitken at Edinburgh. There were lots of people who did it, but Southwell gave it the name.

Now, here was the idea. Do you know who Maxwell's demons are? Okay. Well, Maxwell's demons normally work in a bottle, but Southwell, you can imagine, was employing Maxwell's demons to do the following thing. You have an elastic structure, and you know how each node in that structure affects its immediate neighbors, to which it's connected directly by beams of various kinds. And you know what kind of load you want to put on the structure, but you don't know how it will deflect under that load. So what you do is you hire Maxwell's Demons to hold every node of the structure in some pre-assigned position, regardless of the forces that that will take. Then you go from node to node in some order, not necessarily the same order all the time, and you tell one of the demons, "Let go." His node relaxes into a position of equilibrium relative to the other nodes, which is very easy to compute. Then you tell the demon, "Okay, hold that." Then you go to another demon and let him relax. Every time a demon lets go and lets his node relax, a certain amount of elastic energy dissipates. Therefore, as you go through this process repeatedly, you're dissipating elastic energy. Well, how much energy can you dissipate? If the structure is stable, there's only a finite amount of energy to go.

So gradually, as you go through this process of relaxation, the structure relaxes into the configuration that it will assume under the loads you have opposed. It's nice that this can happen because, you see, otherwise you would have something like as many as six equations per node to solve. In a reasonably sized structure, that's a lot of linear equations to solve, or linearized equations, I should say. And we didn't have computers big enough to do that, so the relaxation method was just ideal for computers of the size that we happened to have. I was studying relaxation methods for diffusion problems. Diffusion problems are, for example, the diffusion of a fluid percolating through some medium, given the pressures or influx of fluids at the boundary of the region. But the same equations work for electrostatic potential, for example in the cathode ray tube, the electrostatic potential of the accelerating electrodes and deflecting electrodes. The same sort of equations worked for a passive electric circuit consisting entirely of resistance and batteries. And one way to model it, which will be good for our purposes, is to model it in terms of a bunch of elastic strings connecting rings, each ring on a post. So you put down some vertical posts. Around each post you drop a ring. You connect the rings by elastic bands to various of their neighbors. You take the outermost rings and put them wherever you want. You may add some weights, or so on, to the rings. And then you say, "Okay, where are all these rings going to go? How will the elastic bands pull?" And, of course, you can solve this by relaxation. But it's a diffusion problem. Unlike the usual elastic problems, where the position of a node depends not merely upon its neighbors, but also on forces transmitted by more distant neighbors, using the intermediate neighbors as sort of a fulcrum.

[Tape 3, Side A]

KAHAN: So in the more general elastic problems, the influence of neighbors more distant than immediate neighbors is significant. In a diffusion problem, only the immediate neighbors matter. I was going to study diffusion problems; I mean, start with something simple. That would have been my master's thesis simply to program it up and see how it works, explain what it does in the light of current understandings, and so on. But I discovered something interesting. You see, David Young's theory worked: for example, the simplest case would be one where you can paint the posts red and black in a checkerboard pattern. The ring around any black post is connected by elastic bands only to red posts, and red connected only to black. That's an example of a simple way in which you can set things up for Young's theory to work. And under those circumstances, there are a few characteristic properties of a network which suffice to tell you how well not only relaxation will work, but also something called successive over relaxation.

Now, let me get back to Southwell and his demons. Suppose you tell the demon to relax, and the node moves in a certain position. For all you know, the next time around, it's just going to move in the same direction. If you believe that, and the sensible thing for you to do is to say, "Tell the demon not to hold the node where it is," but rather to say, "Go half again as far as you went," for instance. So that's over relaxing by a factor of a half, depending on how you look at it, or one and a half, depending on how you want to look at it. But there's a certain factor, you see. Go further by this factor, and then hold it. Now, if you do that, that's called successive over relaxation. If, on the other hand, you said, "No, no, don't go all the way. Just go half way." That would be successive under relaxation; so you can parameterize how much under or over relaxation you want. That's an overrelaxation parameter. And what David Young showed was that, if you overrelax by the right amount, you can speed up the convergence of the process by orders of magnitude. That was astonishing. All you have to do is choose the right amount of overrelaxation. Now, as you increase the amount of overrelaxation, the speed of convergence would rise and rise and would finally rise very steeply. And then, if you exceed that critical amount, the rate of convergence would simply slope off. So, getting the right amount of relaxation was important, and what I found was you didn't need the red-black property. All you needed was a diffusion problem. A diffusion problem is one where each ring exerts the same force on its neighbors as the neighbor does on it. There's a certain reciprocity, so that produces what's called a Stieltjes Matrix, this symmetry. All you really needed was that, it seemed. I didn't know why, but that's all you needed. And then you'd get the same behavior. You increase the over relaxation parameter, and the speed increases. Maybe not quite as sharply as in Young's case. And then you get to this critical spot, and after that it decreases.

HAIGH: And you discovered this by programming it and experimenting?

KAHAN: Yes, absolutely.

HAIGH: Was this a novel way of doing mathematics at that point?

KAHAN: No, Gauss used to do experiments, too. Mathematics is not something that springs full-blown out of a brain. You observe certain interesting regularities or irregularities or whatever it is, and you wonder, Is that the way it has to be? And I observed this behavior and, to be honest, I had lots of computing time in which to observe it.

HAIGH: Was this particular discovery something that someone plausibly could have made without a programmable computer, or would it have just taken way too long?

KAHAN: I don't see how this particular discovery could have been made in that way, although I can see how somebody might have speculated. They might have looked at David Young's thesis and said, "What's essential?" In David Young's thesis, it was the combinatorial structure that was essential. It didn't have to be a diffusion problem; it just had to be this combinatorial structure. If somebody's going to generalize it to a general diffusion problem, he might have speculated that something like that might work, because it worked for David Young. It might have, and I think that's what I did. But then to see whether it works or not, I don't know what you could do other than program a computer. The theory that would explain this stuff existed, but we didn't know it. It just wouldn't have occurred to anyone, without looking up a certain paper by Stein-Rosenberg, that any of this stuff could work. Even if they did look at the paper, it would have taken an awfully long time for the penny to drop—because it sure took a long time for mine.

But, you see, something else was happening. After I'd written my master's thesis, and while I was still working towards and explanation of this stuff (because I was beginning to see why it happened), I got another job from Joe Kates. Now, this was for the DEW line, the Distant Early Warning line. The idea was to build radar stations in the Arctic that would detect Russian bombers coming across the polar regions in time to launch interceptors. But it was cold up there, and it was very, very hard to do construction there, especially if you have to wear mittens all the time. A Greek architect, whose name I have mercifully forgotten, had come up with a scheme that used aluminum tubing with flattened ends and holes punched in them, and aluminum cylinders with slots cut in them in sort of a turn-screw arrangement, so that you could assemble a structure resembling, vaguely, the geodetic dome that you've seen associated with Buckminster Fuller's name. But you could assemble this thing out of aluminum parts, and all the aluminum parts could be handled by mittens. So, this looked like a way to build the platform and the dome, although you couldn't build all the dome that way because the radar has to get out. But you can build part of it. That would house the radar sets up there in the Arctic, but there was a question: would it withstand snow loads? After all, when the snow comes, it comes. You get a lot of it. It's actually very dry in the arctic, but every now and then you get an awful lot of snow in a blizzard. Also, of course, you get these winds, so there was the question: will this structure withstand the loads that are going to be placed upon it?

Joe knew that I was doing experiments with the solution of vast numbers of linear equations, and this problem reduced to the solution of a vast number of linear equations. But we took advantage of symmetries in the structure in order to get the number of equations down below 256, because that was all the memory we had! Yes, I could read the coefficients for the equations off the drum, but I had to keep the unknowns in memory. There didn't seem to be any other way to do it. I wrote a program to do that, and it used over relaxation, because I had discovered that that worked for diffusion programs. It worked for some elastic problems, and it seemed to be working for this problem, too. It was essentially a shell problem, so over relaxation could be expected to work sometimes. But it was going very slowly. Convergence was slow, and we were spending (well, Joe was spending) $400 an hour. That was the standard rate for the computers of the era. Practically, it was a constant of nature. Every computer costs $400 an hour. Well, they went up to $600 an hour, maybe, later. That's inflation for you. And Joe would watch as I would set into the keys the over relaxation parameter, watching in binary. We learned to read binary. There's no point in reading decimal. It takes time to convert it. We learned to read binary, and I was watching to see how the residuals were settling down and getting smaller. But they weren't getting smaller fast enough. I would increase the over relaxation parameter, and they would seem to go down. But if I increased the over relaxation parameter too much, they would go down. Then, they'd come back again and oscillate and so on. It was taking me a long time to figure out was going on. Finally, I said to Joe, "Look Joe, you've been complaining about this." Every now and then, he'd get impatient. Every now and then he would just reach over me and flip the keys himself. I'd say, "Joe, what've you done?" And at first there'd be this drop in the residual, and then they'd come back again with a vengeance. Finally, he learned to leave me alone. But the day came when I said, "Joe, this thing is not converging. I suspect that this structure is unstable." That's the only thing that would explain this type of behavior, at least in my understanding. And Joe said, "No, no. This guy is a well-known architect. It couldn't be." But two weeks after, we learned that they'd built it, and it had collapsed under a snow roof. Well, I got paid anyway. Joe got paid. It really was unstable.

Why did I think it was unstable? Well, in 1955, as a consequence of my thinking about the diffusion problem, the Laplace difference equation that Gotlieb had set me onto, I'd come up with my own ideas about a theory which extended Southwell's ideas and turned them into something mathematically more interesting. I wrote it up as a little paper, and I'm sure that paper exists somewhere in my files. God know where it is. But I wrote the paper. It was ready to be submitted for publication. I felt really proud of myself. I went to the library, and I chanced to open an Italian journal, which had an English language article in it by Alexandre Ostrowski. And there were my results. He was the author of a number of books, publishing papers since 1919, and he had done it. The same stuff, the same elegance was all there, and it was already published in this Italian journal. In fact, it'd been published several months before that. I suppose I could have been crestfallen. That certainly was my first reaction. "Oh, hell," you know, "I've been scooped." But then I looked up this guy, and I discovered he's been publishing since 1919. I've caught up with him!

So after that, I didn't feel so bad; Ostrowski was a notable figure. He published a lot of stuff, including things that overlapped into my area of interest. And indeed, he even came and gave a talk at Toronto once about how to solve polynomial equations. I remember that I questioned what he had done. I said, "You know, do you realize that in the first step of your algorithm, you will incur rounding errors that do more damage to the roots of the equation than everything else you're going to do?" And he was offended. He felt that if I'd come up with a thing like that at his place, that they would have failed me immediately, and I would've been out of the program. But I was right. He shifted the origin, and an origin shift in a polynomial done that way does exactly the damage I described. In fact, Pete Stewart once wrote a paper to that effect, some years after this incident. So Ostrowski and I had, perhaps, the occasional uneasy moment in our relationship, but he was on the whole a good guy. He was very competent. He's written books on nonlinear equation solving, and I've got them. I think well of him. But the main thing was that, as a graduate student, I had to realize not so much that I had been scooped, but that I was catching up. In a little while, he and I would be not equals, but peers. So that softened the blow. I mentioned that because, occasionally, when I have a graduate student around who feels badly about getting scooped, I try to give them the same stuff. "Don't worry about it too much. You're catching up."

Anyhow, it was on the basis of that work, that I felt that I understood the problem that Joe had presented to me well enough to believe that it was not behaving the way it should and that therefore the structure was unstable, that that would be the natural consequence. Because for an unstable structure, you could prove that you weren't going to get convergence. Well, in order to do that work, I had to run for long periods of time on the computer; that was part of what gave me the experience, the confidence, and earned the trust of others around that, yes, I could be left with a computer. So I would run the computer on my diffusion problems. Sheila would come in to baby-sit the computer while I punched up more tapes or took a nap, running it from Friday night to Monday morning. I had programmed the computer so that if it crashed or encountered something weird, it would squeal. So from time to time, Sheila might come and find me and say, "The computer is squealing." And then I'd have to go reboot it and find out what was wrong. It was a marvelous time.

I tell you, being a graduate student was something marvelous. I had to teach some courses. One of the courses I had to teach, Math 3X, that was a course that I had taken in my third year and done badly in. Remember the third year—that was a tough year. But this time I really understood the material well, and I was able to present it well. And the students who took that course would visit occasionally afterward, or we'd cross paths afterward. And they'd say that that was a great course. It prepared them well for their subsequent careers in physics, chemistry, and engineering. It was a course about what are called divs, grads, and curls. Gauss' divergence theorem, Stoke's theorem—these are interesting because they are the first instances when students will encounter things for which there isn't a simple formula. For example, Gauss' divergence theorem says, if there is a fluid in some region, and it has reasonable properties, properties that you'd normally expect for a fluid, then the amount of fluid being created in the interior, integrated over the whole interior, is equal to the amount that's escaping from the boundary. It's a very simple intuitive thing, but when it's explained in terms of mathematical symbols, it says, the integral is the normal component of the flow around the boundary is equal to the integral of the divergence over the interior. So you've got these formulas, you see. And you can't give a value to any of these formulas into the theorem because the theorem is valid for any reasonable shape, as long as it's not too crinkling on the boundary. Surfaces in three-dimensional space can be really horrible, but assuming you have a nice smooth surface the way you're accustomed to, like a potato, this theorem holds. This was the same thing for Stoke's theorem, and this is the first time in the undergraduate syllabus where you get to manipulate these entities, knowing what they mean physically, but you don't know what they actually are numerically. So it has to be presented with a lot of pictures, and that's what I did.

And I feel pleased to this day that the students who took that class felt that they had been prepared better than other students who had taken a similar class elsewhere or from others. They understood what the things said, and a very important aspect of this was that I did not hit the abstraction first. I didn't give them the general case first. The first cases were the examples, getting ever more interesting. The examples are perfectly general, but they look specific enough, definite enough, that people can relate these examples to preexisting intuition about the nature of things and then generalize from the examples to the general mathematical theorem. Then they realize, oh yes, it looks like an abstract theorem with dibs, grads, and curls, whatever the hell they are. I mean, what's a curl? But when you could see it, when you could visualize it and visualize it in more than one way, that's also important. You can see what a curl is in terms of vortices, but you want to visualize it, also, in terms of a general circulation and a flow when you integrate these things. You want to look at these things in a number of different ways in order to get a rounded intuition about it. Then you go on to abstractions, but that's in a later course when you start talking about differential forms and stuff like that. And to this day, I still teach it that way. Some of my colleagues teach the most general case at the outset, and I think it's like bludgeoning a student. I mean, you're hitting them with the abstraction first. Don't you understand what an abstraction is? It's something in common among diverse experiences, and if you don't have the experiences, you can't have the abstraction. But in any event… All right. Sorry. I have the occasional little war with my colleagues.

HAIGH: As you did this during your graduate days, you were discovering that teaching was something that you enjoyed in itself?

KAHAN: Oh, I'd always been good at explaining things, most of the time; but, as I said, sometimes more explanation than you want. And you're getting that now, you see. Yes, I think it was just a natural outgrowth. My parents spoke English poorly, initially, and they depended upon me to help them with their English. When I started going to school, five years old, when we could have supper together (which was not every night), they would want to know what I had done in school, and they would want me to help them with their English. In particular, if I learned things about language, I would relate it to them. So I was explaining things to my parents starting when I was five or six, somewhere in there. It was a part of my job, so to speak, and I'm sure that influenced many aspect of my personality later. My mother actually benefited substantially from that. Her English improved very significantly; my father, not so much.

My father was more of a geometrical thinker. He could lay out patterns. He could visualize how patterns go on cloth, so you put them on there to minimize the amount of waste material. He had a certain intuitive thing. When I would watch the way he would figure out geometrical problems, I was appalled. It was not at all the way geometry had been taught to me, but he had his own ways of doing it. So as you've gathered, I respected my parents' intelligence, although my mother was a difficult person to love. My mother and my younger brother spent the last ten years of their lives not on speaking terms with each other. How that can happen, I do not know. I mean, it's certainly possible to dislike someone, but to dislike someone where you refuse utterly to speak with them—especially a close relative—that seems hard for me to swallow. But that'll give you some idea of what my mother was like. My father was different. He was a real pussycat. I remember when we had some disagreement. I don't remember what the disagreement was. And he laid down the law, but he laid it down in a different way than you might expect. He said, "I am not a German general. You are not a German soldier. I don't give orders and, if I did, you wouldn't take them. We may, therefore, disagree on occasion, and if we have to agree to disagree, well, someone is going to learn who is right and who is wrong the hard way. But I will be insulted if you pretend to agree or pretend to assent when you don't." And I've said the same thing to my sons. My father was not what you may call a lax disciplinarian, nothing of the sort. There were some things that were right, and there were some things that were wrong. And he was going to do what was right, and he expected me to do what was right. But if, after we had discussed it, we disagreed about what is right or what is wrong or what is to do, all right, so we disagree. But don't pretend that you agree when you don't. And often I would want not to do what he said I should do, but I would feel that I had to do it anyway. Then his statement was very clear. If you're going to do it at all, do it well. Either do it well or don't do it, simple enough. That is counsel of perfection, you understand, and it's not always a good idea. But that was the way I was brought up.

My father gave me other advice, too. He gave me advice that I realized later he must have gotten from Napoleon—although the two could not possibly have known each other, personally. Napoleon's advice was if you want something done right, you must do it yourself, and that was also my father's advice. That was the only bad advice I got from him because instead of Napoleon's advice, really what you have to do is frame what needs to be done in such a fashion that you can live satisfactorily with the way people are going to do it. Otherwise, you'd have to do everything yourself, wouldn't you? Well, unfortunately, I learned to live my father's advice before it dawned on me that it wasn't such a good idea. So I've got these unfortunate habits. And that's what Aristotle said: "You are what you do habitually."

HAIGH: So where it becomes relevant, you may want to give an example later in the story of where you've done that, of taking Napoleon's advice where you shouldn't have done. If there's a point in the story where it becomes natural to illustrate that.

KAHAN: I see. Well, what we've got then is a situation where I have the computer to myself for a long time, and I could run experiments. I could become evermore convinced about what was going on. And I found a paper by Stein-Rosenberg in the *Journal of the London Mathematics Society*. I forget, I think it was 1948, maybe, thereabouts. [P. Stein and R. L. Rosenberg, On the Solution of Linear Simultaneous Equations by Iteration, *J. London Math. Soc*. (1948) s1-23(2): 111-118]. And there were a couple of other papers that I found in some of the Russian literature. I learned to read Russian one letter at a time, and I would read Italian painfully with a dictionary, and even Romanian. And French was easy for me to read because I'd taken it in high school. And I had to pass an exam in German, so I spent eight days during which Sheila and I spoke only German. She had taken German in high school. So we spoke only German, and I read only German for eight days. Then I passed the exam. I never learned to love German. In fact, I still dislike it, but at that time, I was able to read at least some German. I've been rusty about that.

And from the Stein-Rosenberg paper, more than the others, it dawned on me why this was happening. It was a lengthy analysis—a horrible, long, complicated analysis, initially—but I finally figured out why the behavior was so similar to what David Young had seen without requiring any combinatorial property at all. But at the same time, I was doing so many other things. I had discovered that rounding errors can be analyzed in floating point arithmetic although, when I started in 1954 (and certainly up to 1956, for example, in that numerical analysis class) the general consensus was that floating-point rounding errors are intractable. Fixed-point rounding errors you can understand, because there's a fixed bound on the rounding error. It's the quantum. The difference between adjacent fixed-point numbers is constant, but in floating-point, the difference between adjacent numbers changes whenever you cross the radix. That made floating-point rounding error analysis seem extremely difficult.

In fact, von Neumann and Goldstein, in their paper on rounding-error analysis of linear equation solving, which they published in 1948, '49, they converted it to a fixed-point problem. [J. Von Neumann, H. Goldstein, Numerical inversion of matrices of high order, Amer. Math. Soc. 53 (1947) 1021-1099]. It wasn't a floating-point error analysis. I think it's fair to say the same for what Turing did in his paper, in *Quarterly of Applied Math* in 1948–49. [Turing, A., "Rounding-off errors in matrix processes", Quart. J. Mech. Appl. Math., 1, 287-308 (1948)]. But Turing dropped a hint in the last or penultimate paragraph of his paper: he said, you know, it's curious that the errors that you commit during Gaussian elimination could be interpreted as if you had perturbed the initial data, and then done the computation without error in the triangular factorization phase. And I read that paper somewhere in, I don't know, 1954, '55, or '56. And the penny dropped.

The same penny dropped for Wallace Givens in 1954. It dropped for Fritz Bauer and Jim Wilkinson in 1957—Bauer in Munich and Wilkinson in England. It turned into what was called *backward error analysis*. It said, when you perform certain computations, not all of them, you can interpret the results that you get as affected by round-off as if you had perturbed the data in end figures initially, done the computation exactly, and then perturbed the final results in end figures. And if you look at it that way (assuming that you can look at it that way, because not all computations are like that), but if you can look at it that way, then you see the effective round-off is the natural effect upon the solution of perturbing the data to the problem. That's just a standard perturbation analysis that one would do of the mathematical problem in its own right, regardless of how you do the computation.

So you see, you divorce the result from the algorithm. Once you can satisfy yourself—assuming it's true, which it isn't always—that there is a satisfactory backward error analysis for your algorithm, you no longer care what the algorithm is! You can tell someone, "Yes, your results are as bad as this, but then, considering that you don't know what your data is, maybe you shouldn't worry about it too much." In any event, if you don't like the answer, don't blame the algorithm. That was the attitude. You know, the attitude is wrong. It's actually wrong, but at that time, it seemed perfectly reasonable. Why is it wrong? Well, it's because when data is uncertain, it is not necessarily uncertain in the same uncorrelated way as rounding errors are uncertain. Sure, your data may be off in the third figure, but usually the perturbations or uncertainties in numerical data presented to a mathematical program are perturbations inherited from uncertainty in physical parameters, which are very much fewer in number than the parameters you present to the subprogram.

You want to do a matrix problem—let's say a hundred by hundred matrix, 10,000 numbers, all of them uncertain. But where did they come from? They may have come, 10,000 numbers, from 300 parameters. That means, then, that you don't have 10,000 independent degrees of freedom; you've only got 300. And if you want to produce an answer, which is as good as the initial data deserves and not merely an accident of your computation, you've got to take the correlation into account. And now people are doing that, and they're doing it in spades. Jim Demmel and I sort of started that kind of stuff, and everybody is trying to do it that way now. Well, almost everybody. But back then, people didn't fully appreciate this notion of correlation, so they took the simple view that I've just enunciated.

And I was figuring that stuff out in 1957 and, at the same time, I was helping my friendly graduate students. Well, these were graduate students, who had taken the course in integral equations with me, or graduate students whom I'd met in the course of slumming around the computer. They had an interesting problem. In fact, they had lots of interesting problems. Here was one of them: it's called the Zeeman effect. The Zeeman effect is what happens to the spectral lines of a molecule which has been excited to radiate. You present it with energy in a suitable form, typically impinging photons. That excites the electrons somewhat, sends them off to different orbitals. After a while they relax back to their initial states, and they emit photons. And the photons have a characteristic wavelength. And if your molecule has certain symmetries, then some of these wavelengths, spectral lines, will be repeated. You know they're repeated, because if you impose a magnetic field that slightly distorts the molecule, what you see is that what looked like one fat spectral line splits and splits into, maybe, three of them. And the amount of the split is proportional to the strength of the magnetic field, at least initially. That's the Zeeman Effect, and that's what some of these guys were doing.

The trouble was that, when they calculated what the spectral lines should be from first principles, and then tried to see whether the separation was what they'd expected. The standard error bounds for the uncertainty and eigenvalues, as a consequence of computation and so on, were inconclusive in the split. And that meant that, although it looked as if they were corroborating their experiments beautifully, the fact was that they really hadn't proved it. And the most important thing that they hadn't proved was that, in fact, that they'd computed as many spectral lines as they thought. You see, the atom is characterized by a differential operator—Schrödinger's equation—which you can linearize. You do all sorts of brutal things to it. It has a spectrum that extends off to infinity. There are infinitely many eigenvalues, and you just want a few of them. So you represent this thing by a discrete system with finitely many degrees of freedom—right there, you're making an approximation. And the fact that it has finitely many degrees of freedom won't give you much very much comfort because that finitely many is a vast number, a huge number, hundreds, thousands, nowadays millions of degrees of freedom instead of infinitely many. How much satisfaction do you get from that, especially on a small machine with 256 words of memory? So you end up approximating only some of the eigenvalues and eigenvectors, and then you wonder two things. One is, I have six eigenvector approximations. How many eigenvalues am I approximating, because none of my approximate eigenvalues is actually an eigenvalue? It's only an approximation, and maybe two of my approximations are actually approximations to the same eigenvalue of the original system. So I think I've got five, but actually I've only got four. I have two grubby approximations to one of them, you see. How do you know? And then, of course, the question arises: how good are these things, anyway?

So I worked out some of that stuff and gave it to my pals to use. On that basis, they then figured that, yes, if they can believe my error estimates, then they have indeed calculated the spectral lines accurately enough to see that they corroborate their experimental results. So they got a thesis. Well, I told you, I was busy with a lot of things, you see? And that was why one day in early March of 1958, Griffith, who was my thesis advisor, and I admired him to the point of idolization because he would listen to me patiently. He would give me reasonably wise advice, even though I was talking about stuff the likes of which he had never seen. When I quoted the stuff that was in the Stein Rosenberg paper and so on, he'd never seen that stuff before, but I was doing my duty as a graduate student. The graduate student's first obligation, of course, is to educate his professors, and I was doing that. And my professor was willing to be educated. Byron A. Griffith, he had been a smoker and had contracted cancer and had lost his larynx. He had one of these little holes, you see, so he had to speak by burping. We wanted to make sure he wasn't obliged to speak too much. After a while, he just gave up teaching because it was too hard. Baxter was the name of his other graduate student. Baxter was supposed to graduate ahead of me because he had started before I did.

HAIGH: And just following up on the last question, so at the time that you were doing this work on error analysis in Toronto before you went to Cambridge, had you been aware that Bauer and Wilkinson were working on the same problems? Or was it something completely independent?

KAHAN: I found out about Givens' report before I left Toronto. I found out about Wilkinson's stuff-- when was it? Yes, that would have been after I finished my thesis. Wilkinson published something in 1957. Bauer published something also in 1957. It was in German. I didn't see it until 1959.

HAIGH: So what did you work on for your thesis? Did the backward error analysis make its way into it?

KAHAN: Baxter used to make fun of me. He mentioned my child bride because I'd gotten married at 21, you see, since we graduated. He felt that was extraordinarily young. There was all sorts of stuff in the world I had missed, you see? I, on the other hand, felt that I was perfectly happy to miss that crap. But once my wife cooked up a batch of muffins, yes, her bran muffins. You didn't really eat them. You sort of inhaled them, and I brought some. Baxter recognized that there was some virtue in having a child bride who could cook. But Baxter's thesis was delayed. He just wasn't as good at it, and he wasn't lucky. So he actually ended up getting his thesis finished, if he did finish it, at least a year after I finished mine. But I finished mine in a rush. Griffith came to me one day early in March, 1956. Well, no, I came to him, and he said, "Kahan, write your thesis. I don't care which of the ten theses you could write that you do write. Write your thesis, and give it to me by May. Or find another thesis advisor."

Hey, this was something I had to take seriously, so Sheila and I went out and bought a typewriter with some mathematical stuff on it. And I worked day and night, and she worked day and night to type it up. It was a thesis about the application of the things we'd gotten from the Stein-Rosenberg theory, plus a lot more. I really elaborated that a lot more in order to show that the behavior Young had observed was characteristic of diffusion problems. It was just once instance in a narrow range of behaviors that I could characterize the behavior without a factor of two in speed, from the same parameters he would have used. And it worked for block overrelaxation and all sorts. It was a very general phenomenon, and I had a marvelous thesis in which I had lovely counterexamples to plausible conjectures. [Kahan, W. *Gauss-Seidel Methods of Solving Large Systems of Linear Equations.* Ph.D. thesis. Toronto, Canada, University of Toronto, 1958]. I remember having a picture in there that looked like the inside of cow's udder to characterize the locus of eigenvalues of a system. It was a very impressive thesis, if I do say so myself, and I was able to finish it on time and defend it in time to get my degree in the end of May, 1958. There is one other incident worth recording.

HAIGH: I'll turn the tape over now.

[Tape 3, Side B]

KAHAN: Okay, one event concerning the delivery of my thesis is worth recording. One of my physics professors was a guy by the name of Barnes. When I was a graduate student, I took a course from him statistical mechanics. I think that Barnes had been regarded as the laziest man on campus until I turned up and stole the palm from him. Well, I became less lazy after Sheila acquiesced to my blandishments in my third year, and then I started to get industrious. But up until that time, I think I had earned the right to be call laziest man on campus, and stolen this from Barnes. Anyway, Barnes was interesting, and I enjoyed his course on statistical mechanics. And he turned up on my thesis committee, as one of the outside examiners from outside the department on what was called the senate oral—the final defense of my thesis, the defense which anyone could attend. They could walk in off the street if they wanted, and they could ask anything they wanted. But there was a committee, and Barnes was on the committee.

I presented my results, leaving everybody suitably snowed and, at the end, the chairman could go around the table, asking people if they had questions. The day before this defense, I had met Barnes walking across the campus from the library, and Barnes had said, "Kahan, I understand your thesis is about solving vast systems of linear equations." I said, "Yes." "Have you ever read a paper by von Koch on infinite determinants?" "No, I hadn't." He said, "Yes, Helge von Koch, et cetera, and gave me the reference.

Well, it seemed like a very good idea to look this up, and I did. And it was in German, so it was a bit painful to read. But it was about infinite determinants in a sense that, as far as I could see, didn't bear upon my thesis at all. It bore upon when you could think of an infinite linear system as solvable, and it seemed to me to be less interesting.

KAHAN: There's a set of conditions necessary and sufficient for the solution of a system of linear equations, even if they're infinite in number, provided the vector space that you're concerned with has appropriate properties that, usually, we have. They rose in context of linear equations, and they provided a better approach to the subject than determinants. But I read the paper, and it said, Helge von Koch in Stockholm. Helge, that looks like a woman's name. A woman mathematician near the end of the 19th century—there couldn't have been many of those! So I started looking up. I looked up biographies and so on. The biographies I could find were all very artful. They had only brief references to Helge von Koch, and none of them betrayed the gender. So I didn't know whether it was a man or a woman and, that night, I asked my father-in-law's dental assistant, who was a German woman from Hamburg. I said, "Is *Helge* a boy's name or a girl's?" And she said, "Oh, it's a girl's name, of course, unless it's one of the northerners, Swedes. They would name a boy something like that." Okay, Helge von Koch in Stockholm, I figured the chances were overwhelming that this was a man, at least on the basis of the information I had time to gather.

Okay, back to this exam! The chairman asked various people to ask questions, and he says, "All right, Barnes, it's your turn." And Barnes says, "Kahan, did you read that paper by von Koch?" I said, "Yes." "Was the author a man or a woman?" And I told him what I just told you. Followed by a lengthy period of silence, the chairman said, "Barnes, you've had yours now. Next." I guess Barnes suspected that I was the sort of person who would be curious about everything, and he was right.

KAHAN: It appeared likely that I was going to get a post-doc because, in 1957 at a conference in Wayne State, I'd presented my thesis results and, apparently, they had received approbation and then some. Gotlieb took me to this meeting in Detroit, at Wayne State, where I had something like 15 minutes to go through 15 slides, lickety-split. And if you want those slides, I might be able to find them. I'll have to mail them to you, because my building is shut down. They're fixing the electric stuff there, so the building is closed. After I presented this material, I think there was one innocuous question. Then it was David Young, Sheila remembers, who stood up and said, "You mean to tell me you've actually proved all that? And it doesn't require Property A?" So I said, "Yes." From that point on, it was clear that I was one of them.

This was something that a lot of them would have liked to have proved and, after the meeting, Dick Varga, who was deep into this subject and who was regarded as the technical master of the field—he was the guy who knew more than anybody else—he sat down beside me, and he said, "Now, show me how you did it." So I showed him, and within a couple of months, it turned up in one of his books. I didn't have to publish my thesis because Varga did it for me, enough of it. It didn't matter.

But you know, at the same meeting, another algorithm altogether called alternating direction method was being presented by guys who were showing that, even in cases where we didn't know why it worked, it worked astonishingly fast. That became, very soon, the method of choice. So I had proved a marvelous thing about an algorithm that everybody had been using until I finally finish proving what it does, and then we were switching to something altogether else. Well, that's the way things go.

HAIGH: So did any of your future work build directly on your thesis topic?

KAHAN: Yes, I continued to explore things for my thesis but, you know, Gene Golub really was more interested in pursuing that direction than I. When he and I met in Cambridge and I realized what he was up to, I said, "Well, you know, he's going to do it, and I don't have to."

HAIGH: There are three more things that you have here on the outline of topics you would like to cover during your time in graduate school. I'll tell you all three of them, and you can choose which order you want to do them in. The first one of those is Chandler Davis and the curriculum modernization and the relationship of mathematics to computing. The second one is just general grad school experiences in relations with faculty and staff, some of which you've talked about, but there may be more there. And the third one is the summer of 1957 at the University of Illinois.

KAHAN: Oh, and there's a fourth, which is the matrix package that I wrote just before departing.

Okay, well let's deal with these things, then. Chandler Davis. Chandler Davis had been, if I remember rightly, at the University of Michigan when the House Un-American Activities Committee asked who had been present at some parties he had attended. He declined to state, so they cited him for contempt of Congress and sent him to jail. I cannot think of anyone less deserving of spending even one night in jail. While he was in jail in the prison system, I think he did write a paper or two.

When he came out, he was not employable at any American university. He got a job for a while as a reviewer for Math Reviews. Somebody got the idea that we could hire him up to Toronto. I think that was 1955. It was at a time when I had an office in a building on Spadina, and it was the guys in that building who were interested in bringing him. But because of his background, we could expect opposition from the conservatives in Parliament and elsewhere, so people were talking about staging demonstrations in his favor and so on. G. de B. [Gilbert de Beauregard] Robinson, he was an old fashioned geometer and he told us to shut up. He said, "Say nothing, and this will go through without anyone noticing. If you write letters to the editor or demonstrations or anything of that sort, you will merely excite opposition." He was right. Chan Davis was appointed. Nobody noticed, and Toronto got this gem.

This guy was a well-educated mathematician, educated in the best places on the best stuff. And what he did was to bring the syllabus for undergraduate education in the math department into the latter half of the 20th century. Of course, in doing so, that meant that courses which had been graduate courses turned into undergraduate courses, so I had to learn these things on my own. I couldn't get academic credit for undergraduate classes, and there were lots of other things to do, so I had to teach myself a lot of that stuff. Okay, that has its good points and its bad points.

Chan Davis is still in Toronto. He has been retired for many years. He still has an office there. He's a friend. I saw him at a recent meeting. He seems to be in pretty good health. His wife is a historian. They used to have to live apart. For a while he was in Toronto, and she was in Berkeley. Then she ended up at Princeton. They used to have to commute back and forth to visit each other on weekends. They have a couple of children who are grown up now and have families of their own, and I've written papers with him. For a while we had offices that were just a couple of doors apart, and I admire him greatly. So that's Chandler David in a nutshell, I guess.

HAIGH: And did this process of modernization do anything to change the place of applied mathematics or to make the computer more familiar to people in the math department?

KAHAN: Well, the departure of Beatty had allowed applied mathematics to regain some notice, but we were so weak in that area that it took a long time to get people. I think one of the applied mathematicians at Toronto for a while was Ren [Renfrey Burnard] Potts, an Australian. He was in this building I mentioned. Another was Jaap Steketee, a Dutchman interested in aerodynamics. Ren Potts was interested in things like how to string magnetic dipoles. How do they line up? There's a name for it, which escapes me for the moment. He also did work on traffic analysis, the analysis of what causes a sort of whipsaw reaction in traffic streams. The Dutchman went back to Delft and married an Englishwoman. They were friends while he was in Toronto. I've asked people from Delft if he's still around, and they don't seem to recognize the name. So I fear the worst. There were a couple of others, but it was hard to keep them because the applied mathematicians were too thin on the ground. It didn't get to a critical mass.

Now, as far as computing was concerned, my degree was in math. There were people who got degrees in other departments, even though they used the computer heavily. The computer may have been the one thing they couldn't do without to get a degree at all. A computer science department was formed in Toronto in 1960, and when I came back from England, I was jointly appointed in computer science and math. That's one of the earliest computer science departments that ever existed. The formation of that department could be credited almost entirely to Kelly Gotlieb, because it was his credibility and I guess I should say poise. He is extremely wise. I think any advice he's given me that I haven't taken I've come to regret; I admire him greatly. He married Phyllis Bloom, that woman I told you about who is a poetess and an author. She has written science fiction novels and so on. She's made a mark in her own right, which completes a circle, so to speak. The relations between mathematics and computer science have been cordial for all the time that I can remember, even before there was a computer science department. I've told you already about a computation I did for Coxeter, the Todd-Coxeter Coset enumeration, which Peter something-or-other couldn't manage. Well, I don't really blame him because I think it was an Englishman by the name of [John] Leech, at Edinburgh, who wrote the best such program, and it was fiendishly clever.

At first, hardly any mathematicians used the computer. I think it was just because they didn't know about it but, gradually, computers penetrated everywhere. When we get back to my return to Toronto, we can talk more about this and talk about Tom Hull.

HAIGH: Okay. So I think you've talked already about many aspects of your relationship with the faculty there and your fellow students. You do have a phrase here: "two economists, and Cervinka."

KAHAN: Oh yes, you're going to love that. As a graduate student, I came to be known as someone who could advise about many things. This was partly because of the things I taught. For instance, for two years I taught the sophomore calculus to people in life sciences. This was a difficult assignment, because people who entered the life sciences had been told by their high-school career advisors that they would have to take only one year of calculus—and here I was, inflicting a second year upon them. The second year was accepted by the life sciences people because agronomists and statisticians who were part of that arena claimed that they had been unable to teach an adequate appreciation of statistics to life sciences people for lack of adequate mathematical background; one year's calculus wasn't enough. And so it had been agreed that they should endure a second year, and I was chosen to teach it because my brother was a biochemist, you see. They figured correctly that I would get lots of interesting problems from him, and I did. But the students were surly. They really felt betrayed, so our relationship was never a comfortable one. I'm sure that I presented a lot of material that would be useful to them in their subsequent careers, but I can't say that they appreciated that material because there was this undercurrent, all the time, of resentment. And unfortunately, it was supported indirectly and, perhaps, unintentionally by the general attitude which I've described to you, that anything you learn about life using mathematics is almost certainly wrong. That is, almost all the biologists really felt that the subject was intrinsically anti-mathematical. Of course this is silly, but it's also true that the kind of mathematics they needed—more than the kind of mathematics that we had on the syllabus for the sake of statistics—would have been what we now call discrete mathematics. And that was not yet in the mathematical syllabus anywhere at all.

Combinatorial mathematics was still regarded as entertainment, you see—recreational mathematics. That did not change in Toronto until the Hungarian Revolution. The Hungarian Revolution saw the departure of a number of Hungarians, including many who had been trained in the combinatorial mathematics of Budapest, where they had been teaching for a long time. Some of that leaked out, so to speak, into Toronto and into other places. But even here, Berkeley, I was the one who selected the first text for the discrete math class. I taught it once. That's all. Most of my colleagues feel that I inflict damage upon undergraduates. The phrase they would use is that I terrorize them. I won't go on with terrorize. I'll accept terrify, but not terrorize. So my colleagues teach that. I think they're still using the fourth edition of the text that I introduced in the second edition. I also advocated that combinatorial mathematics be made compulsory in the math undergraduate syllabus. It had been made compulsory in the CS undergraduate syllabus, but I said it should be compulsory there. The math department has hired some very good combinatorialists, and I'm perfectly happy to feel vindicated without having to compete for recognition or any of that sort. I mean, it is taken for granted now that combinatorial mathematics is a respectable area, even though I'm bad at it. I mean, when I have a combinatorial problem, I have to grit my teeth and work my way through it even though I know that I'm awful. It takes me a long, long time to deal with any combinatorial problem that may arise incidental to what I'm doing, but I feel strongly that the center of gravity of mathematical thought and innovation and development is moving towards the combinatorial. When I run the Putnam problem practice seminar, I do my best to get interesting combinatorial-oriented problems. And it usually pays off, because there are usually combinatorial problems on the actual exam.

But remember it was very different back in 1956, '57—utterly, utterly different, just very different mental attitudes. So I think the closest relationships between math and computer science were actually between statisticians and computing, because the statisticians were the ones who actually had to do the numerical computation. Later, the guys interested in partial differential equations got into the act because, of course, numerical solutions of partial differential equations might be the only way for them to visualize what they want. But that didn't happen until it became possible to visualize things, you seem, and when were we visualizing? Well, you had to get decent screens and so on. I think it's fair to say that we were well into the 1980s before people could display the solutions to partial differential equations on machines that academics could afford. Okay, a matrix package.

HAIGH: The thing that you have here before that is summer 1957 at the University of Illinois.

KAHAN: Oh, yes. Okay. The University of Toronto knew that FERUT would have to be replaced, but by what? The University of Illinois had a fast computer. It was one of the parallel computers. All computers are parallel now, so it doesn't matter. That is to say, they run their data on buses, 32, 64, 128 bits at a time. But the Ferranti machine was a serial machine. What that meant was that the bits come along one wire one after another in the sequence separated by suitable gaps—and, of course, parallel machines are faster. They're also more expensive. We figured at Toronto that we were going to need a parallel machine, because that was the way things were going. The University of Illinois was contemplating the design of a new ILLIAC computer, what had been called the ILLIAC II, and I was sent in the summer of '57 to be a sort of advanced liaison. But it was very clear, not only to me but to others, that there wasn't going to be an ILLIAC II, and the reason was David Muller.

David Muller had an idea for what he called asynchronous logic, what would now be called self-timed logic. The idea is that, instead of having a clock signal distributed over the computer and synchronizing all the logic, what happens is that the logic functions have auxiliary circuitry that says when they've settled down. That "they've-settled-down" signal then gets passed on to subsequent logical modules, which then incorporate that "I've settled down" to "my input has settled down," to "my output has settled down," and so on. Now that means that, in principle, the logic can go faster because it doesn't have to wait to be synchronized and principled. Of course, in practice, you have this completion signal that has to be generated, and that's not a simple matter. Usually it's in the critical path. So in fact, we haven't had all that many self-time logical circuits. There have been some. There's a really good self-time divider by…was it Feller or Williams? I can't remember; That makes sense, because the divider circulates little things within itself. It's a short data path, but otherwise, I think it's fair to say that self-time circuitry is relatively rare.

HAIGH: Yes, so was Muller a faculty member?

KAHAN: He was a faculty member there, yes. This is the David Muller whose name is attached to a zero-finding process that uses inverse interpolation of parabolas, so there's Muller's method. That's the same David Muller. And this is a perfectly decent guy, a reasonable person. I liked him, and it's just that what had seemed at first like a really good idea turned out to have its impracticalities. It needed too damn many transistors to begin with and now, when transistors are cheap, it needs too damn many wires. There's too much delay, so we aren't going to see that. Mind you, distributing the clock signal is a major headache on current architectures, especially since, of course, it consumes a lot of power. So the first thing you do when you're not going to use a certain logical unit is you turn off the clock so it won't burn up power in that area. But Muller's idea was, in some respects, ahead of its time, and in other respects, not practicable. It wasn't that it was impractical. It was just not practicable; it took a while to figure that out. I could see it coming, and I didn't have to report back to Toronto because it was perfectly obvious to everybody. There wasn't going to be an ILLIAC II. So the University of Toronto ordered an IBM 650, which was the next machine.

While I was in Illinois, I was nominally under the supervision of Don Gillies. Don Gillies was a Canadian who came to the University of Illinois, and I liked him, too. He was a very decent guy. He was very considerate of the graduate students there. And he assigned me a simple task: figure out what the Simplex algorithm was. Why did it work? What was it doing? The Simplex algorithm was Danzig's contribution to optimization theory for linear programming. Danzig died just very recently. But it was presented in terms of tableaus, tables of things that you update this and modify that and overwrite this, and so on. I was baffled. I couldn't understand what the hell they were getting at. Why are they doing these things?

Well, I got back to Toronto and it dawned on me: imagine an ant crawling on a lump of coal in the dark, trying to reach the highest point. Well, it crawls up the face until it comes to an edge. Then it crawls along the edge until it comes to a vertex. At the vertex, it has to figure out which edge is aimed upwards. Then it goes along that. What can it do? That's what the Simplex algorithm does. Degeneracy is a terrible, terrible problem. What does *degeneracy* mean? Oh, it simply means that there is an edge in the lump of coal that's horizontal so, if it's the highest edge, anywhere will do on that edge. You see, when you're looking at it geometrically, it's all perfectly straightforward. Well, the subject isn't that simple, because there are questions of complementarity, duality, and stuff like that that make the subject much more interesting. But in any event, I made no progress while I was there on understanding Danzig's algorithm, but I did get to meet a number of friends.

At that time, there was no air conditioning except in the computer room (and there was certainly no room to bunk down there!) and in the Student Union. And all the couches in the Student Union were occupied by people trying to sleep because it was so hot elsewhere that many of us couldn't. But I rode a bicycle, and I learned how to sleep with windows open at night and so on and it wasn't quite so bad. Sheila visited me for a week or so. It was hell, but people were willing to drive a hundred miles, I think it was Danville, to see a tree that had not been planted purposefully by man. You could go out there and you could look to the horizon and not see any bump that was not erected by man. Corn and beans, corn and beans, they rotated the crops. Beans were typically soybeans. Then there were fields that were called succotash fields, because these were fields in which either the corn or the beans were sprouting, unwanted, because of the previous year's seeding. The seeds had sprouted late. It was awful, but I met some guys there whose company I enjoyed. I did get to learn some things. I programmed the ILLIAC a little bit, and I met Gene Golub.

That was very brief. He welcomed me, but then he went off to Bell Labs for the rest of the semester. So he and I spent just a week there. That meeting was crucial because I came to know Gene Golub before he encountered Abe Taub. Abe Taub was…what shall I call him? Abe Taub was like mistletoe on people's personalities. It was sort of a parasitic element. To give you an example, Don Gillies threw a party for graduate students very shortly after I arrived, and I had already done a number of things that were interesting at Toronto. Apparently word of that had gotten to Illinois, so that's why they accepted me, I guess. And at the party, Abe Taub came up to me. He said, "Well, Kahan, what great theorem have you proved today?" And my instinctive response was, "I'm not sure you'd understand it," which was the right response. So Abe never bothered me again. Abe ended up in Berkeley. He came to Berkeley, I think, in 1964, which was four or five years before I did. His office was just a few doors down from mine, so we did get to know one another. He even lent me some of the volumes of von Neumann's Collected Works, of which Abe was an editor; so it wasn't as if our relationship was brittle or anything like that. I mention this because Abe was not kind to his graduate students. One of his graduates students was Charles W. Gear. Now, I think Gear is at NEC in Princeton, now.

HAIGH: He recently retired.

KAHAN: Yes. Well, he's entitled to. Gear had a reputation then as the only sane one amongst everyone. I mean, faculty and graduate students, he was the only sane person there.

[Tape 4, Side A].

KAHAN: Yes, Charles Gear, his reputation was that he was sane, as most of us weren't, that he was very competent, and Taub was his thesis advisor. And I remember often hearing Taub berating Gear in his office, and Gear giving back as good as he got. He wasn't going to take any of that sort of crap lying down. But other students at times were intimidated. Golub was one of Taub's students. I didn't know it in 1957, but I found out in '59, what a horribly destructive effect Taub's personality had had on his graduate students. Gene Golub had been a cheerful fellow, renowned for puns and for a *joie de vivre*, I suppose. At least, if he wasn't like that all the way through, at least he put on a good show. And when I saw him in '59, two years later, he had finished his thesis under Taub; it was a respectable thesis, I don't think it was a brilliant thesis; it certainly wasn't as astonishing as mine, but it was a perfectly respectable thesis. And Gene had been destroyed by the encounter with Taub. Apparently Taub had berated him, preyed upon his weaknesses. And it's hard for me to forgive that. That was a very important aspect of my visit in 1957 to have encountered things that I didn't fully appreciate until later.

HAIGH: So would you say that Golub's personality bounced back over the years?

KAHAN: No. I don't think it ever recovered fully. And that's why I find it hard to forgive Taub, although he's been dead for some years. Gene has done a lot of good things, and he does a lot of traveling and interacts with many other people, so there were lots of joint papers, and inspires that kind of work. Gene and I have co-authored a paper that was in 1964; that was an eventful spring. A SHARE meeting, after which I stayed over for about a week to finish a paper with Gene at Stanford, but we'll get to that later.

I think there is one item, though, that deserves mention: apropos software packages. When we got the 650, we had to change a lot of our ideas about programming. The 650 was a drum machine. Every instruction was 10 digits: a two-digit OP code, a four-digit address of the operand, the implicit destination being one of the registers whose addresses were in the 8,000 series. We had a disk with 2,000 words. There might have been disks with as many as 4,000 words.

HAIGH: The drum?

KAHAN: Drum. Yes. We did get, very soon, an upgrade, which put 60 words of core memory into a machine. They had addresses of, I believe, of a 6,000 range. And it was a shock. Every instruction had a second address, which said this is where you should go to find the next instruction. And it's very peculiar in its instruction set. Now, I had not, before then, gotten much involved in software packages for other folks to use. I was mainly concerned with my thesis or if any particular job at hand. I did write a really super-fast divide program for the Ferranti machine, and I think I mentioned a super-fast input program. But hardly anybody else could use that super-fast input program, because it was dangerous. It would tear tapes. On the IBM 650, things were different. We started to have a situation with program libraries. One of the things that was signally missing was a way to do matrix computations. The languages that we had would have been either the Symbolic Optimum Assembly Program (SOAP), or there was something called FORTRANS, which was an early adaptation of FORTRAN for the 650—not very good. And anything to do with matrices was a terrible pain.

So I wrote a matrix interpreter. What this did was to interpret program statements that looked like SOAP statements. They had op codes and they had operands but, now, sometimes two operands. Instead of having operand and destination for the next instruction, they would have two operands. And each operand now was to be interpreted as the name, actually the address, of a matrix on the disk. So it wasn't the actual address of anything, it was just the address, which the interpreter would interpret as saying it was this array on the disk. And when you declared these arrays you didn't process a certain amount of indirect addressing. The operations would be the obvious ones: add, subtract, matrix multiply, divide (that's linear equation solving). I think I had an eigenvalue program that used Jacobi's method; I had transpose, and I had a program for solving the normal equations. And it wasn't the best way of solving the least squares problem, not by any means. But that's a story for later, perhaps.

All right. What this meant was that you would jump to the interpreter, which would interpret the subsequent instructions after the jump as matrix operations, after which it would interpret a certain instruction as leave the interpreter, and then you could continue with ordinary things. So you could bury the matrix operations in the midst of your code. It would read like ordinary code but, from a certain point on, you were doing it to matrices instead of scalars, that's all. Furthermore, we had sixty words in the core, so what I managed to do was put the inner loop into ten of those words, and the other fifty words would match the fifty-word circumference of the problem. It meant that, now, my matrix operations could be timed for one of the operands on the drum to pick up operands as they came by, because the other was in the core—no delay. So the matrix operations went very fast for an IBM 650. And I remember finishing all the documentation, the programs, the tests, and handing over to the decks the whole shebang. I think it was on a Thursday before the Sunday when we were to take the train…with Friday to pack. And when I got to Cambridge, I sent them my phone number with the understanding that if anything came up, they could phone me. But I was never phoned.

Well, of course that could mean a couple of things. It could mean that my programs were just not perfect or it could mean that nobody was using it. Well, when I got back to Toronto the 650 had already been replaced by the 7090. And what I discovered was that there was a 650 emulator running on the 7090 with a very high duty cycle. It had been used a lot and I thought, "What are they using it for?" They're using it to run my matrix codes, especially the statistics department and the guys doing the statistics, or actually the statistical users, psychologists, economists, sociologists and so on. Whoever was using standard statistical programs apparently had gotten used to using my package on the 650, and was still running it emulated on the 7090. It ran a little bit faster than it did on the 650. So they did use it. I think that was my first package. My first packaged software—my first delivery.

HAIGH: So in this case you were quite remote from the user experience?

KAHAN: I was in Cambridge. So I had no idea what was happening. Nobody told me.

HAIGH: Okay, let's conclude Session 2.

Session 3 begins in the home of Dr. William Kahan. Saturday, August 6, 2005.

KAHAN: I think that when I was taught programming, initially by Trixie—Beatrice Worsley—and by…I hope that I've remembered correctly the name of Cecilia Popplewell, because I'm really bad with women's names. At that time I found out that the majority of programmers were women. Now that was partly because of British practice during the Second World War that women were operators of the equipment that the British used to decrypt German-encrypted communications; there was a terrible shortage of men. And that persisted for a few years—just a few—after the war. By 1960, I think things had changed to pretty much the picture that we see now. And that was largely because the majority of computers with any substantial computing power were being used in connection with defense. Consequently, it was men hiring men and it just didn't occur to them to hire women, despite the fact that women had worked during the war producing ships and aircraft and stuff like that.

HAIGH: Many computers by the late 1950s were being used in business.

KAHAN: They were, but the computers used in business were relatively smaller computers. I did write programs for some of the business computers, but the one with which I was mostly associated was the IBM 7094, for example. The 7090 would have been overkill for business.

HAIGH: And I've also seen the suggestion that one of the reasons that women were found as programmers on the ENIAC, was because of wartime distribution of roles, but also that applied mathematics, and in particular routine calculations, had traditionally been an open area to women compared with other parts--

KAHAN: Sure, there were whole rooms filled with women working at desk calculators. It was a job considered beneath men in the U.S. In Britain, there were still lots of men who used calculators, but in the U.S.—in actuarial offices, for example—there would be some man who would figure out what had to be done and then would hand it to the girls to do the computations on these huge paper sheets and so on. Yes, I remember seeing those rooms.

HAIGH: So were there many women studying mathematics when you were at Toronto?

KAHAN: There were more women enrolled in math, physics, and chemistry *pro rata*—that is, the ratio of women to men was higher when I entered college in 1950 than among students throughout the 1960s, '70s, and '80s. Late in the '90s I began to see a rise in the number of female undergraduates in math and computer science and in electrical engineering, but still not very much in electrical engineering. Unfortunately, too small a proportion of them go on to graduate school. But that proportion is being increased, slowly and painfully. I've tried all kinds of devices to get women to participate in the Putnam exams—never works. A couple of years ago, I enlisted the help of one of our Miller Fellows, who comes from Romania and got her Ph.D. in the U.S. She had scored high on Olympiads while in the U.S. I think she was the first woman to stand in the top five in the Putnam exams, and I made sure her position was prominent. I made sure that there was no hint that she was subordinate to me. I hoped that that would attract women, but it didn't. Last year I was on sabbatical. The Putnam problem practice seminar was conducted by this woman and another who is a Morrey professor: also very, very sharp and extremely competent at problem solving, because she had scored high on Olympiads. Both of them now are running the problem sessions. Still only one or two women, despite the fact that the undergraduate advisors encourage women who appear to be talented math undergraduates, and also some others, to participate—but they don't. And I don't know what I'm going to do this year to change that. It's possible that women associate competition… Just the fact that it is a Putnam competition, a competition in which if boys are involved women would rather not participate. I just don't know.

HAIGH: And I understand that you had a particular story that you would like to tell about one of your fellow Toronto students.

KAHAN: Yes, Harriet. She was far and away the brightest in math, physics, and chemistry that year. That includes David Gauthier, whom I've mentioned, but who left to go into philosophy instead. Harriet was just good at everything. We would struggle with some of the math problems that we were given in our first year. She always seemed not only to have the solution to every problem, but elegant, lovely, crafted solutions. She aced the exams and she did marvelously in physics, which was the direction in which she went. When we graduated in 1954, she already had some kind of scholarship that took her off to Cambridge, where she got her Ph.D. in physics. And I got my Ph.D. in math in Toronto and then went off to Cambridge. Now people seem to think that Canada was a small island, because as soon as I would say that I was Canadian they would say, "Do you know Harriet Thompson?" And I would say, "Sure." They would tell me Harriet stories. Well, for one thing she was charming. And so when she and her fellow graduate students went off to have lunch in places that had traditionally admitted only men and there she was, this woman sticking out like a sore thumb, the various flunkies there would say this is only for men. And Harriet would say why and chip away. Her friends would say, "Oh come on, she is one of us," and stuff like that. So she broke down those barriers simply by acting as if they didn't exist. And she was very sweet and good-natured about it. It was extremely difficult to say no to her. So when I got to Cambridge, she had already left. She had gotten her Ph.D. the few months before and she had already gone. But she had left her mark all over the place. And that is why I got to expect it when someone said, "Oh you aren't American, are you?" And I said, "No, I'm Canadian," and then they'd say, "Oh! You must know Harriet." And they would tell me another Harriet story.

Harriet met a South African that shared her interest in dramatics and stage, Colin Cryer. And they went back to South Africa because she felt, I'm told, that she wanted to assist in rolling back apartheid. I think she raised a family there, but for some reason, which I have no idea of, her marriage failed. In any event, she had to come back to Canada to care for some sickly relatives, it may have been a parent or an aunt, an aged parent. And so she had to take a position at a lesser Canadian university teaching physics in order to be near where this person needing care lived. And that is the last I have heard of her, and what little I know has actually come to me through Joe, who was my lab partner in the optics labs and a high school chum. He had kept in touch with her, as I had not. She was so superb, and to think that, ultimately, care-giving responsibilities fell to her because-- well, that put quite a crimp in her career.

HAIGH: So that would be a good place to turn the tape over.

[Tape 4, Side B]

HAIGH: How was it that you went to Cambridge for two years?[4]

KAHAN: When I went to England, well, the original plan had been to work with Douglas Hartree. Hartree had visited Toronto in 1957, and he was interested in calculations of molecular orbitals. There's a name for the method, Hartree-Fock's scheme for calculating these—this was for calculating in the spectra of molecules, and you would do so for a number of hypothetical configurations of the molecule in order to discover which configuration had the least energy. So that would be the natural configuration for the molecule to take. You wouldn't necessarily know in advance where everything was, but you'd learn something from X-ray diffraction. Hartree-Fock is a pretty heavy-duty calculation, even though it's a linear eigenvalue problem which you linearize and solve iteratively. And, you know, they're still doing it. In fact, that's what Clemens Roothaan was doing lots of. But Hartree died. He died early in '58 after arrangements had been initiated.

HAIGH: And I think you implied that there was some type of government fellowship established program?

KAHAN: Yes, it was a government-funded post-doc, funded by the Canadian Government. I had to apply for it. And I think that I got it on the strength of the results that I delivered at this meeting at Wayne State. On that strength it was easy to get letters of recommendation. I don't know exactly, and how does one ever know. I mean, you fill out a form and send in some stuff, and I never saw the letters. So I don't know what they said. But it was a very comfortable $1,500, which seemed like a pittance in Canada but turned out to be quite substantial in the UK at that time.

HAIGH: Had you considered going straight to a faculty position?

[4] Some of the material in this section occurred earlier in the interview but has been reordered for to enhance the flow of the interview.

KAHAN: I had considered going straight to a job. But I had been given advice that a post-doc was a good thing to do, and I give that same advice now to graduate students—advice from both Griffith and Kelly Gotlieb. And I really wanted to get away and do something different. I would have been willing to go to France, but there wasn't really that much to go to in France. There just wasn't that much computing. I might have been willing to go to Germany; it's hard to say. But the chance of working with Hartree really attracted me and, I think, that was the reason beyond all others. Regardless of everything else, Hartree had visited Toronto and I had been so impressed that I thought, I would really like some of that guy to rub off on me. But it didn't. But we certainly didn't suffer at Cambridge—those were the best two years of our lives. But we'll get to that at another meeting.

HAIGH: Perhaps we should pick up the story with your arrival in Cambridge.

KAHAN: Well, when we arrived in Cambridge, we thought we should get a car. And I wanted a Morris Minor. But the waiting list for the Morris Minor was two years, and our plan was to be there for two years because this post-doctoral fellowship was renewable. So we ended up buying an imported car, a Volkswagen. I remember still that we came into the showroom, these two scruffy people. And the salesperson wouldn't even look our way and thought these two people are just gawking. But when it became clear that not only were we going to buy a car but also we were going to pay for it on the spot, oh, the obsequiousness was something to marvel at! Yes, we bought a car in London and then Beetled up to Cambridge and found a place to live.

I had to report as a graduate student—they are called "research students." My Ph.D. from Toronto was just from the colonies, and wasn't recognized by Cambridge. So I had to enroll in Johns College as a research student, and I had to accept a tutor from the college and a thesis advisor, who was J. C. P. Miller. And I wasn't terribly concerned about such questions of status and so on, and I didn't really care so I figured some money is going to go—think of it as a charitable contribution—to Cambridge University and its institutions. That turned out well for me. Not the part in college, where I had to get special permission from the tutor to skip meals in hall and eat meals at home with my wife. Married students weren't something they were really prepared for at that time. Of course it is commonplace now; at that time they just weren't set up for it. And anyway, I wouldn't have wanted to eat in hall because, occasionally, people there would be somewhat rowdy. I remember, on occasion, people getting up and clomping across the table in their boots to get to a bench in another place, and, well, the whole thing my mother would say was not cultured and refined. So I did make the acquaintance of several people there, and I formed a good deal of respect for the talents that were assembled there.

Later, when I came to found out what happened to the people whose talents I rated most highly, I found that they had hit various glass ceilings in British industry and academia, and had consequently emigrated to Canada, preponderantly the U.S., and to a lesser extent Australia. And I have mentioned that in my thing about U-boats and the war, and British captains of industry. Military captains credited themselves too highly for having won the war and then felt that this gave them the right to permanent establishment to British hierarchy in industry and academia and thereby choked off paths of advancement for talented, younger people.

Well, those talented, younger people simply walked away, carrying the technology with them. The Brits, like Americans and Canadians, imagined that technology is something that you can put in inventory. You can have software you own; it's on your shelf. You have these patents and other intellectual property. They don't appreciate properly that intellectual property is of no use to you unless it also exists in someone's head. And more than merely memorize it, it has to exist in someone’s head as a flourishing, growing entity. And if that growing entity walks out the door and takes your property with it, notwithstanding noncompete contracts and so on… the fact is that, when you lose your talented people you have lost a hell of a lot more than a body. But maybe that is a story that we should return to another time. But in my thing, I have said a consequence of that, the British technological lead enjoyed at the end of the second World War, was dissipated. So England has become, I think, I call it a quaint island destination for tourists.

It was fun. I would discover various things, and I would sometimes go downstairs to ask David Wheeler about these things. David Wheeler clearly understood the state of affairs of the subject. If I would ask him questions about it, provided I could get him to answer at all, they were helpful answers and indicated what sort of stuff he had known beforehand and so on. I can’t say that he and I were friends because, although I admired him and I liked him, he was extremely reticent. I think it went well beyond English reserve. He was just like that.

But I also traveled down to Teddington. I would take the train to London and then change at-- I think it was Waterloo Station for a train up to Teddington and visit Wilkinson at the national physical labs, and I would show him some of my stuff. Most of the time, he’d look at my stuff and say, “Yes, let me just look.” He’d reach up, way up and pull out an envelope, blow on it. Then pull out something and say, “Yes, I think I’ve got that,” as he leafed through his notes. “Have you got this, too?” So it was that kind of relationship. I enjoyed those visits very much. Usually I would have lunch with Wilkinson and some of the other guys there, because I’d try to come down after the rush hour.

But actually, it seemed to me that, in London, there was no such thing as “after the rush hour.” I remember going down steps in Waterloo station while almost everybody else was coming up. I’m six feet tall. Most Englishmen weren’t, and they’d all be wearing these bowler hats. So I’m looking down at this sea of bowler hats bobbing up and down as they come up the stairs while I’m trying to find my way down. But I’d get there usually early enough that we would then have lunch and then have some more conversation in the afternoon, and I’d get back before closing time.

So I had a chance not only to speak to Wilkinson, but also to speak to people who had been also, like Wilkinson, contemporaries of Turing. Now, Turing had killed himself in 1954, and I’d always wondered about that. And whenever I’d ask people who knew him why this had happened, they would change the subject. I found out later, a lot later, first that homosexuality existed, because it hadn’t crossed my mind, and second, that he was homosexual. Third, that instead of confining his homosexual interests to, say, King’s College or someplace like that, where people would have turned a blind eye to it, he became involved with a working class man, and that was considered a class violation. I think his friends did not stand up for him as they should have and, in consequence, acquiesced to degrading treatment. He had to take all sorts of drugs and stuff. He lost his security clearances. I learned all about this afterwards. They wouldn’t tell me at the time, anything about this, and it casts a bit of appall over my appreciation of those people, that they let this happen to the guy. But we have to appreciate that homosexuality was not just a criminal thing then. It was something that people thought of as dirty, ugly, and repulsive, so it must have been a great strain for his friends. I don’t know what I would have done under like circumstances. I realize now that some of my teachers and friends were homosexuals. I realize that now long, long after the fact, and I find it difficult now to understand why people get so exercised about the subject. Why do we care about people’s behavior in bed if they are both adults and not related by blood? It all seems very self-limiting. But, all right, so I never understood it. So that was one of the things that I now remember as having been one of the persistent mysteries. But other than that, as far as technical things were concerned, it was a marvelous time.

HAIGH: And what was the state of the computing at Cambridge at that time? Was the team that built the EDSAC still intact?

KAHAN: No, EDSAC had been dismantled. EDSAC 2 was the machine they had, and was an initial version of the microcode which they were revising while I was there. Many an evening saw David Wheeler actually threading the wires through the core in order to reprogram the microcode. I got to use that machine—not for building software that would go into packages or anything; that was not my concern at the time. I was concerned with extensions beyond my thesis, and exploring the behavior of various iterative methods. I wanted both to explain mathematically and to enhance in a programming sense the efficiency of iterative methods, not merely the ones that I explored in my thesis. I came to understand the conjugate gradient method. So I was just writing programs to run experiments. And I shared an office with I think it was Jamie Cran; he was an American. Nick Capon was an Australian and went back to Melbourne and, ultimately, was running the computer center there. Gene Golub turned up in 1959 from the University of Illinois. And there was someone whose name I can't retrieve, although he was in some ways the most interesting—he was from South Africa. His thesis interests were in things like stellar evolution, and the solution of partial differential equations in domains whose boundaries are moving. That creates some really interesting problems in characterizing the shapes of these regions, because the motion of the boundary can lead to little loops, occlusions, inclusions, and things like that. I feel terrible that I can't remember this guy's name, even though I think he was the one who was somewhat the most competent.

HAIGH: So was there an office in the computing lab?

KAHAN: Yes, soon as we went in the door we could look down at EDSAC.

Well, there were also diploma students. These were students who were proceeding not towards a Ph.D., but towards a somewhat lesser degree at the time. Among them was Mike Powell, and he is now, at least until recently [now retired], he was chairman of the NAMTP, or something like that, Numerical Analysis Group, Department of Applied Mathematics and Theoretical Physics, I think that's what all the initials stand for. Mike Powell was one of the students in a course I gave there. I gave the first course in error analysis at Cambridge. I even managed to get some question on the tripos. Of course hardly any students ventured to try it. But he was in that class, and subsequently made his name in optimization methods. He was also the editor of the proceedings of a conference in Birmingham in 1986, where that document that I'm also trying to find on what are called the elementary inverse transcendental functions for complex variables. This means things like arccos and arcsine and logarithm and the square root—these are all the inverse functions of various entire functions. Functions that are handled everywhere. And they post a number of interesting challenges. He edited the proceedings, and he had to retype my document because I sent it to him printed off this horrible printer that I had…this dot-matrix printer. We should talk about this sometime later. I hope I can find the computer files so I can print you off a copy.[5]

[5] HAIGH: Is there a reason you have stuck with dot-matrix technology?

KAHAN: Well, the reason that I became a captive—it is a familiar reason and I am not the only one who suffers from this. As a historian you would have doubtless many cases. When the IBM PC first came out, I got one. They wanted to know what I thought of it and I sent them a long list of interesting observations, which subsequently led to a program called Paranoia. Paranoia is one of the earliest programs of a kind that Jim Cody describes as MACHAR. It's a program that diagnoses the arithmetic of the machine it is running on. And I produced Paranoia in order to explain to the people at IBM as to why the arithmetic on the earliest IBM PCs was so awful. They did change it again. That is the second time I have gotten IBM to change their arithmetic! But there I had this IBM PC--

HAIGH: Was that the arithmetic in the BASIC that was built into the PC?

KAHAN: Yes, that is correct. It was the floating-point arithmetic in the ROM of the basic. It wasn't the arithmetic in the coprocessor, because when these machines came out they had a socket for the coprocessor. But the coprocessor cost initially $150, so there weren't that many people who bought them. But when the price came down to a little bit less than $100, suddenly all the sockets got filled. But that is a story for another day because that is a story with Bill Gates, Jr., in it.

So I had this IBM PC, that must have been 1981–'82. And I had IBM's printer. But I also had a dot-matrix printer, more or less upward compatible from that, which Intel had given me initially, like a processor developer system, which I did some of the work that led to the 8087. Then later, Intel actually gave me a more advanced computer—an Intel 302. But what I discovered was that I could download a font into a slightly modified version of the IBM dot-matrix printer and, later, into the one that Intel gave me. What you had to do was to get some chips to plug into sockets on the Epson (and therefore IBM) printer. Now, the significance of downloaded fonts to mathematics is this: at that time there was no significant software available for personal computer people to do bitmapped graphics. Everything was character-driven. Your screen is full of icons—we didn't have that. The screen was just simply strings of text characters. These text characters were initially generated in read-only memories associated with the display device. So the output from the computer would select a character; one byte was generally sufficient to select which character you wanted to display. Then the display hardware would draw from its read-only memory—the strings of bits it needed in order to display a line of text. And a dot-matrix printer imitated that, and so you would send a single byte or character to the dot matrix printer, and it would get printed on the page. But you would have to form a line first. It's a whole line of text, and then the dot matrix printer would form the necessary code to drive these little pins to make the characters on the page, in a way pretty reminiscent of raster scanning. It wasn't until the Macintosh came out that there was an inexpensive way to get graphics—bitmapped graphics, where every pixel on the screen was accessible—and you could make pictures on the screen and mathematical characters and different fonts. So, initially, all you had was what is called a sans serif font on the screen.

HAIGH: And then Hercules came out a graphics card that drove the standard IBM mono monitor that bitmapped the screen.

KAHAN: Yes, that's right. But in order to make that work you had to buy the Hercules card and you had to get a display that would accept the output from the Hercules card. It wasn't just any old display; you couldn't drive IBM's monochrome display adaptor from a Hercules card, if I remember rightly.

HAIGH: I think you could. I think that's why the card was so popular. But that's hardly germane. So you stuck with the dot-matrix printer because of the downloadable fonts that you had been using?

KAHAN: Yes, I fashioned downloadable fonts so that I could-- I'm trying to think of the Hercules card. I remember inquiring about that, attempted to get a Hercules card, and I was told that I would get the card and have to get a different display. I had a different CRT display.

HAIGH: That was the case with upgrading to CGA or EGA graphics but, I believe, the attractiveness of Hercules was that you only had to change the card if you had the mono green screen.

KAHAN: Oh, that green screen? I don't remember that. In any event, it certainly didn't drive the printer. I never did get the display to show the same characters as were displayed on the printer. Because I started as soon as I got the PC, I realized that I wanted to make notes for my classes. So I cooked up a special font that included mathematical characters and fragments from which I could construct mathematical characters by a backspace overtype. Well, it's obvious like with the Greek letter theta, you type a 0, then a backspace, and then type a minus sign and then you have the letter theta. If, instead of a minus sign you type a slash, you have the Greek letter phi. I managed to get alpha, beta, gamma, delta, epsilon and capital delta and all sorts of things on there to go onto the printer. Subsequently, ROMs turned up which let the printer interpret all 256 characters in the fonts. They had some of the Greek letters, but they didn't have all the ones that I wanted, so I had to prepare fonts for some of the other things that I wanted.

Once I had that, it meant that I could use the word processor, which was called Easy Writer. I could use the word processor to inject appropriate characters into the text as I typed it, then get Easy Writer to convert these characters into the command sequences that would allow me to use these special characters on the printer. So what I saw on the screen would be-- for example, if I wanted a superscript. Now there was a superscript capability in the printer, but if I wanted a superscript, I would type a character, which came on the screen as an up arrow, and then I would type the letters I wanted in superscript. Then I would type a character that would look like a double exclamation mark on the screen, and Easy Writer would automatically translate the up arrow into the command sequence for "switch to superscript mode." And the double exclamation mark would get translated into "stop superscript or subscript mode." It was a down arrow for subscripts and various characters for square root sign, and so on and so forth.

So I would type up these manuscripts, and I know that what I read on the screen wasn't exactly right, but I could see in my mind's eye what it would turn into and, indeed, it did turn into that. And it did it at full speed—not at graphics speed, but at full character speed, which was a lot faster than the graphics mode. The graphics mode would bang out a row of dots per byte and that took seven to nine, depending on how you wanted to do it, seven to nine bytes per character. And it was slow because you couldn't assemble a whole line and then sweep through it in one shot—they had to done one character at a time. I was doing it in draft mode. That meant I could get up early, prepare my notes, print out a copy, go to the copy machine, copy a number of copies for my class, and turn up in the morning and hand out the sheets. And the same thing with the solutions. I've got 300 floppy diskettes with that sort of stuff.

So, of course I became a captive. Then along comes EGA displays and various other things which turned everything topsy-turvy, but made it possible to get beautifully typeset stuff. But here I was a captive of this old, clever technology. Well, I've been trying to convert these things working on my favorite old Macintosh, using a word processor that, when I first started using it, was called FrameMaker. It was considered the best for this purpose. Alas, Adobe bought FrameMaker, and then announced that in May of this year that they were going to discontinue support for FrameMaker on Macintoshes.

And just to make my life more interesting, FrameMaker on Windows uses Microsoft's fonts, instead of Macintosh's Adobe fonts. The fonts have the same name, but different character sets. Well, you have probably seen occasions when someone sends you what is supposed to be a text file, and it comes out with weird-looking characters. Certainly the ones you sent me had weird-looking characters, and I had to run them through conversion. I've got all these Macintosh files and I can't manipulate them using the same word processor, FrameMaker, on my Windows machines, because the special characters that I use, some of them are different. What's more, Windows has a very inconvenient way of incorporating these characters. On a Macintosh, all you have to do is press the appropriate combination of option or control key and the character, and you get what you want. You can figure out what it is because they have a keyboard image for any particular font and if you bring down that keyboard image and press the appropriate keys, you see what the characters are and that is what turns up on your manuscript and so it is easy to type. On Microsoft's thing, you have to do a click and drag operation. Click-control-c to capture the character from a table, and then you have to do a drag in your manuscript and drop it in. So instead of just typing along you have to go through all this mousing stuff.

HAIGH: If you press ALT and then type the four-digit code for the character, you can insert it that way.

KAHAN: Tried it; it doesn't work on FrameMaker. If it does, it is working on a version of FrameMaker that I haven't had or haven't explored. But, I did my best to look into the help thing to find out what to do. I am going to look that up and see if that works now.

HAIGH: It used to work in-- I know we used it in PageMaker, I think it was supported at the Windows level.

HAIGH: So did you have a sense, at that point, of Cambridge as a place where exciting work on computing was taking place?

KAHAN: Oh yes! For example, the initial work by Crick and Watson was done on EDSAC 2. There were people from all sorts of departments floating through. But I was interested in a relatively modest number of things. There were a number of different difficult mathematical questions, and I wanted to work on those and do computing to see whether experiments would shed light on the questions—and used EDSAC 2 for that purpose. I did give a course there. I did participate in numerical analytical discussions with my friends in the office and with others and with J. C. P. Miller.

I'd like to talk about J. C. P. Miller now. He had achieved fame in connection with the British Association Mathematical Tables Committee. Now this was the equivalent, in Britain, of WPA effort in the U.S. to keep mathematicians employed producing tables. Of course, with the advent of computers, the tables fell rapidly out of disfavor: although there was a woman who was finishing off a project on the dilogarithm family of functions. It was a project to print up tables, and may have been the last set of tables of a serious mathematical family that were ever printed up. Printing tables became futile after that because memories got bigger. The EDSAC 2 didn't have a terribly big memory. It might have had 2000 words, or some small number like that. It had a tape and, if you really wanted to do things, you had to learn how to bring things down from the tape and bring things up on, perhaps, a different tape and so on. So that really slowed down my progress.

HAIGH: Did it have floating-point hardware?

KAHAN: Yes.

HAIGH: So, was that the first machine with floating-point you had used?

KAHAN: No. The IBM 650 was the first machine that had decimal floating-point hardware. That is the first one I used. A contemporary 704, I think, had floating-point hardware.

HAIGH: Yes, I believe that 701 didn't and the 704 did. And core memory—those were the two big advantages of the 704.

KAHAN: Oh yes! We salivated over core memory. EDSAC had core memory, but it didn't have a lot of it. It had two core memories—one was the read-only memory. That was the one that David Wheeler was running wires through cores in various ways, and that was where the microprogram sat. Maurice Wilkes may have had the initial ideas for microprograms, but it seemed clear to me at the time that Wheeler was the one that was actually doing the microprogramming. EDSAC 2 had an instruction set that included instructions for the elementary transcendental functions, for linear equation solving, for a Runge-Kutta method for

---

KAHAN: Well, it used to be that you typed ALT and then a three-decimal digit code on PCs in order to get peculiar characters. That still works, but when I try to do things like that for FrameMaker's fonts obtained from Windows, it didn't function.

HAIGH: So I will move this discussion in the transcript over to a later discussion of PCs to keep things together. So back to Cambridge.

KAHAN: Of course at Cambridge we had nothing of the sort. And in Cambridge, if you wanted to type a manuscript you just typed it. First you wrote it and then you typed it. There was no word-processing capability there.

solving differential equations. I think it also had a rudimentary program to solve a linear equation and with one real unknown. And I don't remember what other things it had, but let's just say that the instructions set for the machine was ample. And the programming language was essentially assembly language. Compared with the Ferranti Manchester Mark I, it seemed remarkably easy to use. But managing the tapes was a real nuisance and, unfortunately, the calculations that I wanted to do involved the transfer of information back and forth with the tapes. That was just too rudimentary and so it slowed me down quite a bit. I think it slowed everybody down quite a bit.

Mind you there was some very clever guys there. There was Peter Swinnerton-Dyer, HPF. He was by far the brightest guy there. He was interested in algebraic number theory. But he seemed to be a polymath and seemed to know a good deal about damn near everything. I know when I consulted him about problems in complex variables it made me feel like an idiot when he acted like it was a perfectly obvious sort of thing. The girls all seemed to have set their caps for him, and he seemed to be indifferent to them. I know he ultimately rose to positions of considerable power and influence, and I think he ended up dispersing funds to various colleges. I imagine everything he did was done with a certain finesse, and his reasons for doing things were very, very hard to contradict. He was really very good. He and another guy whose name I have forgotten—I think it was David Barron—came up with various schemes for doing computations involving the tapes, for sorting and transposing matrices and things like that. But a lot of it was misplaced effort.

The evolution of computers was nowhere near so fast then as it is now. I think what they were doing was trying to fit size 13 feet into size 10 shoes, since larger computers were obviously on the way. And with larger computers, many of their techniques would have been rendered if not unnecessary, then at least less valuable. But the most important thing was to get rid of those damn tapes and go to disks, because tapes were accessible only sequentially. If you wrote stuff on a tape in one straight line, that was the only way you could get at it. And the disks gave us an approximation to random access. So it was the fact that disks were visible on the horizon when I was at Cambridge that made me feel that they were in a way wasting their ingenuity.

HAIGH: I think the first machine with disks was the IBM RAMAC.

KAHAN: Well, that was the first disk drive system.

HAIGH: Yes, obviously drums had been around longer than that. And I guess just about the time that you left Cambridge, drives would have become commercially available for a variety of general purpose machines.

KAHAN: I think they had some drum project there, too. I can't remember accurately. Drums gave you something approximating random access, but the trouble with drums was that were so hideously expensive—you had to have one head per track. You had to switch electronically the heads. For some reason, also, the drums seemed to be less reliable. Certainly on the Ferranti machine, they switched from a nickel alloy-plated drum to a ferrite drum while I was there. But it still never seemed to be all that reliable. The IBM 650 had a reliable drum, but they were using (if I remember rightly) error-detecting correcting codes in order to make it reliable. So the electronics worked—they compensated for the hardware, if my memory serves me rightly. But when disks came, and error-detecting correcting was incorporated into disks and so on, it was a vast improvement in random access to bulk memory.

HAIGH: So I think you wanted to say something about J. C. P. Miller?

KAHAN: Yes, okay. So he had become interested in number theoretic calculations. And there were occasions when we would see him seated like a happy Buddha, looking through his results, some of which were still being punched on paper tape that he was getting off the EDSAC machine. Well, he was my thesis advisor, and he loved to tell me stories about the good old days. And I learned all sorts of marvelous tricks and techniques from him. In his office he had at least one Brunsviga—this is a crank-turning machine.

Well, I felt he was a kindred spirit partly because of that. When I was working on my thesis, I had taken home a Madas, which is a Swiss electromechanical machine. I had taken it home because it was futile to leave it to anyone else. Somehow they would manage to jam it, and then I was the only one apparently who knew how to un-jam it. So I might as well take it home. But this machine made a terrible racket and wake up the whole house when I was using it. So I had experience with that machine and a number of others—Marchant, Frieden, and a long list of names that aren't important now. But that was something I had in common that I had with J. C. P. Miller. That is, he and I were acquainted with a wide range of electromechanical machines. And that was just the beginning of a long and fruitful relationship.

When Hartree died, he left his numerical analysis courses in a somewhat untidy condition, and no one to teach them. So Maurice Wilkes thought that he could teach them. Wilkes was a man who found it easy to dismiss people whose talents were not so well rounded as he thought his were. Wilkes seemed to have a poor opinion of J. C. P. Miller. Well, he didn't seem so much to disparage him as treat him as a person of not very great consequence. So it wasn't to Miller but to Wilkes that the task fell to teach the numerical analysis course that Hartree had given. And I sat in on that course and it was a disaster. It became clear not just to me, but to the whole class, that Wilkes was in over his head. He didn't understand numerical analysis; that was very clear. He certainly didn't understand the way Hartree did. To say that he didn't understand is not to say that he was stupid—far from it. There are lots of people who would teach numerical analysis classes without understanding the material. To them, the material would seem like algebra: you are manipulating formulas. The whole point is to rearrange the formulas into such a form as to leave to a sequence of assignment statements, and these assignment statements carried out, in many cases repeatedly, in loops will ultimately build a solution to your problem. But of course that is the way it may look at first, but that is not what the subject is really about. Hartree, I think, had understood this. And from I could glean that percolated from Hartree's notes through Wilkes, showed me that he had a certain insight into this stuff that Wilkes lacked. And so it was a dreadful course.

I used to ask questions because I didn't know I wasn't supposed to. On one occasion, Wilkes attempted to explain the Laplace Difference Equation (which had been my master's thesis subject) and he wanted to give a probabilistic explanation that he was drawing from Hartree's notes. And of course I knew what that probabilistic approach was, and Wilkes didn't. So what he did was to say let's suppose that we consider the solution to this partial differential equation converted to a partial difference equation and consider the solution to be a probability. Already he had started off on the wrong foot. Up came my hand and I asked, "How do you cope with negative values for the solution if they are probabilities?" And he said, "Oh, you just add something to the boundary values so that makes all the solutions, positive—and then you just divide by some suitable constant so the numbers go between zero to one." Obviously he was quite off the mark—and all that really did was make matters worse. And after a while of him explaining how you interpret these things as probabilities it became clear that, as he went through the notes, he was finding out that there were things that didn't stick together properly.

And the class was fidgeting a little bit and so on. Finally the end of the hour came, and Wilkes said, "I'm going to have to look into this and deal with it later. Kahan, would you come to my office?"

Let me interpose here before we go into the discussion in his office, the solution is not a probability. It is rather that the solution at any point in the interior of a region is a positively weighted average of the values on the boundary. And you can interpret these positive weights as probabilities that are random walk—starting from a given boundary point will end up at this particular point in the interior after a reasonable number of steps, or the probability that it will pass through this point. Therefore you can interpret the computation as probabilistic in terms of random walks. But the solution is not a probability! It is rather that you can express the Green's Function's value as probabilities. And Wilkes in preparing his class must have overlooked that.

So I came to Wilkes's office and he said, "Kahan, as much as I appreciate your questions, I would prefer that they be conveyed in private, henceforth." And I said, "If that is the way you want it, I will do it that way." And I did. And then he added, "This is an elementary course in numerical analysis." And I said, "Well, there is a difference between elementary and superficial." A dark cloud crossed his face and that was the end of the conversation and I was dismissed.

HAIGH: Now is that the story you mentioned that you both tell about each other?

KAHAN: We both tell exactly the same story. For him it is a story about me, and for me it's a story about him. Now I have to say my relations with Wilkes were not unpleasant. He was hospitable and made it possible for me to do my work, although he didn't see a hell of a lot coming out of it for him. So I had no war with Wilkes. I think if anyone should have a war with Wilkes it should be David Wheeler who, I think, labored in the shade—underappreciated—because Wilkes felt that it was sort of work-for-hire, and he should take the credit. I don't think he gave David Wheeler enough credit until he retired and then David Wheeler became a professor. Wheeler was a genius. Wilkes was a very competent fellow. I think that's the difference.

[Tape 5, Side A]

HAIGH: Actually I have a question.

KAHAN: I want to return to J. C. P. Miller.

HAIGH: It's related to that. So he was officially your thesis advisor. Did you actually have to produce a thesis in the end?

KAHAN: No, we knew it was a farce—an administrative convenience. I was an administrative anomaly, and this was the way of dealing with it. J. C. P. Miller was not a genius. He was not broadly competent in many, many things. He had come up through a path of numerical computation in days when it was a very dreary subject; it did not attract the world's best minds. He had flourished in that environment and had done very good work and become very experienced. He became experienced in a kind of deviousness that the others lacked. Yes, Wheeler was ingenious; Wilkes was organized; but J. C. P. was a devious old man with all sorts of sly tricks. And I enjoyed that aspect of our relationship. He was happy to tell me of the tricks that he had used, and I was happy to hear about them. They influenced what I did in the same ways as Kates had been extraordinary devious in his programs that were marvelous marvels of intricacy.

J. C. P. Miller was a man who saw something in numerical computation that the others had not seen, and it was getting lost. When people did their computations by hand, be it on a slide rule or desk machine or whatever, they would be producing a sequence of numbers according to a formula. But every attempt was made to give these numbers some kind of meaning, so you could get a sense that maybe there was something weird about these numbers—something that you should distrust. So much labor went into producing these numbers that if they turned out to be wrong (either because you made a slip in key pressing, or because rounding errors were accumulating in a dangerous way, or maybe because this is not the computation you should not have started on at all) you would want to learn about that soon because the labor involved was so great. So that meant you would try to understand the numbers at various stages of the computation when you planned it so that you could get some idea of what they ought to be.

Von Neumann thought that everybody should do this and it was perfectly natural. Von Neumann felt that, in the course of the computation, that these intermediate quantities had physical meaning and that was why you don't need floating point. Any man who understands the computation could tell immediately when his numbers had gone astray, or he can know what scale factors are needed. He knows in advance what the range of his numbers should be because they have some sort of physical significance, albeit very, very indirect. The people who did numerical computations by hand had to have this kind of feeling in order not to waste more time than, inevitably, they had to. If something went astray, you would like to know about it sooner rather than later. And that sense was being lost as people did their computations, and especially as they were doing it in higher-level languages. They would write out formulas, and they would have lost any incentive to imbue their formulas with some kind of meaning derived from the initial problem—so they would lose some sense what kind of numbers they deserved to have, what kinds of relationships they deserved to preserve.

In some computer science communities they talk about invariants, and so on. Miller was talking about checksums and other schemes that you would incorporate, in which a small fraction of extra work was being done in order to satisfy yourself that the computation up to this far was not blighted by mistakes. Miller was teaching me about how, when you have computed a table of numbers, some of them might be a little bit wrong. And so you then construct a difference table, which can be done relatively cheaply, especially since a National Cash Register Company had built a differencing engine. It was a tabulating machine designed so that, if you entered a column of numbers, it would automatically print out not only the numbers but also a difference table. So then you could look down the difference table, and if you saw a rough spot, you'd say, "This rough spot in higher-order differences points to the possibility of an error at this place in my initial table, so I should perhaps re-compute that or check it or maybe just smooth it out." With respect to the other values, that might be a cheap and easy way to do it—though not altogether honest.

So I learned about a lot of these things that were dying elsewhere, and that's why I treasured my interactions with J. C. P. Miller. He was a decent old fellow, maybe a bit doddering at times. Some people would laugh at him because he seemed to be so interested in these number theories of computations that others did not necessarily appreciate. I thought of him as a decent, honest, workman and I was happy to learn everything I could from him.

Occasionally we would drive over to Oxford to attend some of the seminars there that would have been in Leslie Fox's establishment. Leslie Fox was a brilliant character—altogether different cloth. He was admirable in an altogether different way. I don't think that he was brilliant, either. He was a competent and knowledgeable man who not only made substantial contributions, but also he had sheltered a number of people who worked with him, for him, under him. So I thought highly of him. He was also very experienced, as was J. C. P. Miller, but Fox was experienced at a somewhat higher level of management and organization, and a broader range of computations, I think. Miller was mainly interested in special functions of mathematical physics because of his experience at the British Association of Tables. Fox was interested in solving differential equations, functional equations of all kinds. He did good things, too, and I learned a few things from him, though our acquaintance was always in passing.

HAIGH: Did Oxford University have its own computer at that time? Was it working with NPL to share something?

KAHAN: I can't remember. I know that they had not just one computer, but they had access to more than one. I don't remember what it was they had; it wasn't their own computer. It was someone else's computer that they had access to. It might have been at Harwell, which is not terribly far. I'm sorry, I just don't remember what computer they were using. I think it was a KDF-something-or-other...

HAIGH: Oh, yes! The KDF-9 I have heard of that. I'm pretty sure that Brian Ford talked about the KDF in his interview. Anyway, I'm sure that will be available from secondary sources.

KAHAN: Yes. Fred Ris worked on his thesis at Oxford, and then went back to work for IBM. And his thesis had to do with integral arithmetic and I think he worked on a KDF-something-or-other.

HAIGH: Did you meet Christopher Strachey at Oxford?

KAHAN: Oh, yes. Well, actually I had met Chris Strachey at other places, too. I had met him before I met him again at Oxford. I think I had met him very briefly at Toronto, because he had come to Toronto with a Manchester machine. I was passing through and I don't think he noticed me. But at Cambridge—he did come up there from time to time. And I think I visited him on a couple of occasions in London, when he was working for English Electric. And, yes, I believe I met him at Oxford also. Although I think his appointment at Oxford was after I left. I don't remember exactly those details. He wasn't appointed at Oxford when I first got to Cambridge. I think his appointment came somewhat later.

When I first got to Cambridge, he was working at English Electric; I went down to meet him partly because I was interested in eigenvalue computations. He was using, I think, triple-precision arithmetic and other things to get around some awful problems which ultimately John Francis…I don't think he solved them completely, but his QR algorithms certainly changed the picture drastically.

I was actually a referee in John Francis's paper I submitted to a computer journal, something like the British Computer Journal. Like the Royal Society doesn't have to call itself the British Royal Society, the *Computer Journal* doesn't have to call itself the *British Computer Journal*. But in any event, John Francis submitted his manuscript, I think in 1959, and I labored over that thing, trying to figure out why it deserved to work. And I delayed its publication because I was trying to figure out why it deserved to work. You could see the proof, but the proof is a very stark and sparse passage through the relationships. You can say, "Well, I can't find a mistake in the proof, so I guess it's okay," but that doesn't explain why the damn thing deserves to work. It was really…it took some years to understand that. Not just for me—it took some years for people who were trying their damndest to understand it and figure it out.

Now I ended up writing papers that were illuminating, and I think I wrote a paper that proved it had global convergence for permission problems, and wrote papers with Parlett. [B. N. Parlett and W. Kahan, *On the convergence of a practical QR algorithm*, in Proc. 4th IFIP Congress, 1968, A.J. H. Morell,Ed. , The Netherlands: North-Holland Publishing Co., 1968, pp. 114-118]. When we finally felt that we were coming to understand it, and we don't understand it fully now, it is still works better than we can explain.

HAIGH: When you first reviewed the paper, did you realize how important the algorithm would be in practice?

KAHAN: Yes. That was why I was so concerned, because if this thing worked the way John Francis's examples worked, it was going to put our whole approach to eigenvalue problems on an altogether different footing. Before that time, there were all sorts of different folks who had come up with all sorts of different algorithms hoping to achieve their little measure of immortality by attaching their name to so-and-so's algorithm. Each of these algorithms worked some of the time and failed most of the time. I think the only algorithm that was reliable all the time was Jacobi's Algorithm for the symmetric eigenproblem. Lanczos' algorithm would fail from time to time. We came to understand that only years later. I met Lanczos a couple of times, too—a delightful little fellow.

The QR algorithm seemed… let's put it this way: you could do it in such a way that seemed it would work in a certain sense all the time, if you're willing to let it be slow. If you want it to go fast, you would have to take certain chances. And under very rare circumstances it might not work properly and you would have to do it again. If you could detect that it wasn't working properly, and detect that soon, well, then that's quite okay and I'll do something else. Which is essentially a policy that we do now. Every now and then we may discover that it's not working the way it should and we have to put in a dodge, and every now and then you discover that the little dodge doesn't work. In very, very rare circumstances you have to dodge again. That is enormously different from algorithms in which you invest a great deal of work. They work slowly and after a while you realize all you've got is dust and dirt, which is the way things used to be before. Chris Strachey had thought he would be able to attack the problem by carrying extraordinarily high precision, which was slow at the time but a reasonable approach. Except despite carrying high precision every now and then it would fail and we wouldn't know why.

I felt that this was important enough that we should understand it because it seemed to me that is why going to be a sea change. Finally it did get published, and it had the effect that I had anticipated. It had the troubling qualities that troubled me and troubled a lot of other people. Well, some people are less bothered by these things than others. I think [Jim] Wilkinson felt that he had understood it well enough, and he wrote it into his book. But Wilkinson was a little bit glib about this; that is to say, although the methods seemed to work almost always, I don't think Wilkinson had a good feeling for why the thing failed when it failed, and how you should recognize it and what you should do about it.

HAIGH: How would you describe Wilkinson, in general?

KAHAN: I thought he was a great guy. I liked him for all kinds of reasons, maybe even having nothing much to do with the interests that we shared. I just thought he was a great person. And he was a raconteur—he had a hollow leg. I'm teetotaler, but I'm not puritanical about it, as you know. I don't think he had a mean bone in his body, and he had good insights into things. When he explained something, it was all laid out there for anyone to see, in most cases. There were just a few places where stuff was a little bit troublesome. His skill at exposition was marvelous, and I tried to emulate it but I couldn't because my prose is Dickensian. It is almost impossible for me to write a short sentence. I think he represented our craft to the world in an elegant way and he was clever about things. I don't think he was in the same class as Swinnerton-Dyer; maybe comparable to Wheeler, in a different way. He crafted things well and did a terrifically good job at the things that he did. What I mean by a good job is that he came to an honest understanding of what was going on, and then he explained it well. He was competent enough at analysis that he could solve, sometimes, very hard problems. If I fancy that, on occasion, I have solved some that were harder than his, they were not a hell of a lot harder. When I would visit him, as I told him, he would sometimes cheerfully reach up to a stack of papers (he had these stacks of papers too, not quite as disorganized as mine), he would blow the dust off and say, "Oh, have you got this one, too?" It was a great privilege and a joy to have dealings with that guy.

HAIGH: So is there anything else you would like to say about your time in Cambridge?

KAHAN: Oh gosh, it was in some respects the best two years of our lives. Sheila has mentioned that she got a couple of jobs; she was a secretary in one of the departments for a while, and was a teacher this elementary school for a while. Because dusk lingers for so long in Britain, especially during the summer, we were lucky we had the two best summers in living memory (so people would tell us). So we would Beetle about in our imported car, and Sheila would choose the itinerary. It had to have a village with thatched roofs, or a great garden, a famous house that could be visited, or a church—with a Norman church on a Roman foundation or whatever it was. So she was interested in art and architecture and gardens, and I was interested in airfields and bridges and railway stations. And we managed to work them all into the itinerary. So we would play as tourists in the neighborhood around Cambridge. Sometimes it would be off to Ely; there's a fairly impressive cathedral there. And I can't remember all the places that we went. In a certain sense, we were enriching each other's lives by exposure to each other's interests. It wasn't that I disparaged her interests; quite to the contrary, I was happy to have a little bit of that world to see, and to appreciate and she was willing to tolerate a little bit of my world, so if I crawled under a bridge she would sometimes come to see. "Goodness, look at the way they have done this—they did it with rivets." I remember on subsequent occasions when we visited England and looked at Brunel's bridge—the one that has the gorgeous suspension bridge, which has not cables, but shackles—over the Tay in Bristol, I think that's the one. It is just a gem. You can approach it from the roadway below and you see it there, lit up. Aside from being a work of art, when you actually get up there and see the bridge, see these heavy lories rumbling across them; I know he didn't have them in mind when he built the bridge, and there it is, still functioning. It is a masterpiece of its kind in so many ways. Then there is Iron Bridge; this was the first bridge built out of iron castings and put together as if it was a wooden bridge, with iron wedges instead of wooden wedges.

So I marvel at these things and Sheila would come along, and we visited some of the airfields. Now Ducksford is now a historical museum but, at that time, it was still an active airfield. It had been a very active airfield during the Second World War. But now they were flying I think they were called Javelins, these delta-winged aircraft, flying with some English Electric things. But every now, and then we would find them flying some of the older aircraft. On one occasion, we drove to a farmer's field and walked for a little while until we clambered up over a berm and sat there watching Princess Margaret handing new colors to the RCAF regiment there, or whatever it was. And in her honor, they were flying an old Spitfire and an old Hurricane. We sat on one side of the airfield and way over there on the other side were the tents, and it was like a garden party!

All those great big hats and various alcoholic drinks, and probably little cucumber sandwiches for all I know, served there. The Hurricane and the Spitfire—and what a thrill to see those!

So, we enjoyed life in Cambridge and got to know interesting people. Even our landlord had been a doctor, and he fancied that he had a system for beating the wheel at Monte Carlo. Since I was a mathematician, he wanted me to accept it, and I tried to explain to him is that the only way to beat the wheel is if it is crooked. But he didn't accept that. He had these papers and so on.

There were some of the folks that I met in college. I've already mentioned that these were guys who I thought had talents that merited a considerable higher rise than they have achieved in Britain, and when I got to know them or meet them for occasion, later they had emigrated, much to England's loss. It was a very poor policy but, unfortunately, a very common policy: failing to understand that we have to make space for the people who are going to take over after us. And we have to provide them opportunities to make mistakes. That is a digression; after all if I am such a great manager, how is it that I am not rich. Right? It's like that old question, "If you're so smart, why aren't you rich?" If I were that rich, you wouldn't be able to see me.

HAIGH: You mentioned that you shared an office with Gene Golub. So you had introduced him yesterday and mentioned the difficulties with his advisor.

KAHAN: Well, yes. His advisor was Abe Taub. Taub ultimately ended up at Berkeley; in fact, he died there. And he mellowed some before he came to Berkeley, but before he came to Berkeley, he had earned a reputation at NYU and at Illinois as a shark. Golub was exactly the kind of person who would suffer, because Golub was insecure. He didn't deserve to be insecure, but he was. I think Taub preyed upon those insecurities. Not planning it, but it was Taub's nature. So when Gene came to Cambridge on post-doc—I don't know why he chose Cambridge—he was a different person. A great deal of the sparkle had been burnt out of him. Unfortunately I didn't help, because I had a tendency to complete his sentences for him, and that made things difficult for Gene. He was not at that time articulate enough to compete with me in finishing sentences. He became more articulate as time passed, but at that time he lacked the self-confidence that he had exhibited when I had first met him at Illinois, and he was hospitable and showed me around. That self-confidence which gradually returned in large measure, but never returned fully. The joyfulness of life never returned.

He was working on iterative methods which, in some respects, were similar to mine, except they weren't as difficult to analyze and, in consequence, some of them persist to this day. There are occasions when those things are still useful because they are easier to analyze, whereas successive over relaxation stuff, if it works, it usually works for reasons that things that can't be explained. I explain them for diffusion problems but we'd like to use them for other problems. Why they might work or not work for other problems is a matter for experiment.

These techniques were being used, if we go ahead a little bit, when I visited IBM in 1967. They were working on what is called *chaotic iteration*. This is a scheme where you do the same sort of thing as I've described in iteration: you tell one of Maxwell's Demons to let go, perhaps hold on again, or push it or pull it or whatever if you want to over or under relax. In 1967, people were doing a lot of work on parallel computations, even though they didn't have parallel computers to practice on. There the question was, let's suppose that you have parallel computers doing this relaxation so that, every now and then, when that node is allowed to relax, this node, also allowed to relax, has the old value instead of the new. So it relaxes according to the old; therefore, it hasn't quite done what you expected to do. Will this converge? And the answer was pretty obvious—that, yes, it would under reasonable assumptions and reasonable restraints on what you do. I didn't regard that as a terribly interesting problem because it seemed to me that the outcome was foreordained. But I mention this simply because people were still working on these things and they are still working on them to this day.

Gene Golub, every now and then, worked on this, too. The question is, how do you know how much to over relax? But I think that other methods are more potent when they can be used and a great deal depends on the memory organization of your computer, as to whether you could use them or not. So I was still doing a little bit of that, but I was more interested in error analysis. I worked through a large number of other problems, many of which never got fully solved and I still have them in envelopes, some which I've handed off to students to see what they can do with them. I wasn't really interested in writing papers; I was more interested in broadening my acquaintance with the field in which I figured I was going to work. I don't mean meeting more people; I mean learning more about other people's computations and learning more about why they work and why they don't, when they don't. And that is one of the reasons why I missed Hartree, because if he had been more around I would have learned more about the computations that were germane to mathematical physics. But in his absence I ended up working on a diversity of computations that didn't go as deeply as I would have gone if Hartree had been there.

HAIGH: So when you returned to Toronto in 1960, had your job already been arranged?

KAHAN: Well, I had a number of choices. I could have applied for jobs at other universities, which is what we would do, nowadays. But if I was going to teach, I felt I was going to teach at Toronto; I had a debt to repay. Not a debt in money terms, but I think you understood the nature of this debt. The other possibility was to work for industry, and I know there was still a job at General Electric. I could see advertisements and positions open in Canada from newspapers in Britain. I would have been amply qualified and eligible to take those jobs. But when I inquired about how things were at Toronto, I was told there was a computer science department being formed, and they had a place there for me. I said "What about math?" and they said it could be a joint appointment in math and the new computer science department. So I saw that I could bring two of my interests together and teach about it. I hadn't realized yet the depth of the damage that had been done in 1957 by the election of Diefenbaker as Prime Minister of Canada. If I had appreciated that I might have made a different decision. I think I should tell you the Diefenbaker story, because it affected my career terribly—even though I didn't know it at the time.[6]

[6] I told you about the "DEW line," this early warning line. The American strategy would be that they would get warning from radar of Russian bombings coming over the polar regions in time to send up their interceptors to shoot the Russian bombers down over our heads in Canada. Canadians weren't altogether happy about that, and so a project was developed in the 1950s to develop Canada's own interceptor, which would have the speed, range, armament, and electronics to intercept the Russian bombers rather farther north—over extremely sparsely inhabited Northwest Territories, for example. Perhaps over Hudson's Bay. That was born the Avro Arrow project. Avro was a British firm, but it had an outpost in Canada; during the Second World War, Canada had built a certain number of aircraft. We had built Mosquito bombers, for example. I had bicycled up as an 11-year-old to watch them testing Mosquito bombers. They were building them at the rate of almost one a day. Sometimes they would test two or three a day in flight because, of course, what you find out from testing is that you have to do something to adjust it or fix it. I don't think any 11-year-old boy could have been more thrilled than I was to lie in the long grass to watch these aircraft taking off and landing over my nose. I cycled up—it was like 15 miles. There was a good deal of production of war material in Canada and then, at the end of the war, Canada's immigration policy was very different from the U.S. The U.S. had a quota policy, but in Canada, if you brought skills deemed by the government to be in short supply, you were admitted and could bring your family, which was the only humane thing to do. So we had a large influx of aerodynamicists and people skilled in stamping and working with sheet aluminum and other alloys with

metallurgists and ceramicists and people who were experts on jet engines. People coming from Britain and Germany and all sorts of places. So high-tech flourished around Toronto. There was the highway built around the city instead of through it.

HAIGH: A bypass?

KAHAN: A bypass, yes. It was Highway 401. And scattered around 401 were these machine shops and sputtering shops and electronic shops and ceramic shops. Almost every other place—it was one of these high tech little establishments, many of them founded by expatriates from Britain and elsewhere. And they developed an aircraft called the Avro Arrow. The Avro Arrow was obviously a decade ahead of everything else. It had much greater range than other aircraft, as it needed to have to cover the distance in Canada. It could carry an enormous armaments load; it had advanced radar electronics; and it had two ferocious jet engines. They were called the Orenda jet engines and they were, at that time certainly the most powerful jet engines in the western world and probably in the whole world. So it had amazing performance and they built three of them.

These things were obviously too expensive for Canadians. The government policy in funding this development was that, once we had gotten this aircraft to work, the Americans would have to buy it because they had nothing like it. So they would end up paying for it in order to put Arrows in their Air Force. It wasn't as if the Americans had not bought foreign-built aircraft because the Canberra jet bomber was indeed bought by the Americans and given a different name. And even now the Marines are flying these vertical take-off jets—the ones that crash from time to time—called Harriers. Another British invention.

But in any event, so this was under the Pearson government—Lester B. Pearson. Liberals. They expected—and everyone expected, including Americans—that they would buy lots of these things and build them under license, and so on. This would cover this costs and them some and then we could afford to have these aircraft ourselves. But we didn't figure on Diefenbaker. He was the leader of the Progressive-Conservative Party in the Prairie Provinces. And he was a populist—exactly the opposite of what you would expect the Progressive-Conservative Party to put forth, because the Progressive-Conservative Party was, first, not progressive; and secondly, they didn't conserve anything except the wealth of the wealthier members in the country. They represented business interests in Toronto and in Montreal, preponderantly. But the Progressive-Conservative Party also had its branches out there in west of Canada, and this guy was very popular there; he became the candidate there for the position of Prime Minister if the party was elected in the next election.

He stumped on a hell-raising, anti-American platform, utterly uncharacteristic of the conservatives. Why anti-American? Well, at that time, Americans were subsidizing their wheat farmers, and that made American wheat cheaper than Canadian wheat. Naturally, we were offended; American wheat was inferior to Canadian wheat. Canadian wheat was harder than the American variety, and because of this it made better, fine flour. That was before people realized that white flour is not the best thing for you. They were consuming it in large quantities. Consequently, Canadian markets were being eroded by an inferior product at a cheaper price. So Canadian farmers in the Prairie Provinces were upset about this. The trade relations between Canada and the U.S. have always been somewhat marred by hypocrisy. For example, there was a reciprocal agreement for clothing. Which meant that a certain amount of Canadian-made clothing could get into the U.S. without being tariffed, as a certain amount of American-made clothing could get into Canada without being tariffed. So, of course, my parents thought that they would be able to sell their dresses to willing American customers in Buffalo and Cleveland and other places. But the American dress manufacturers and their union (which was essentially a company union) had an under-the-table deal with the American customs people who would have to admit this stuff. But what they did was delay it. They only had to delay it by two weeks to miss the season because, of course, summer dresses have to be shipped in early spring in order to get through the pipeline. There were all sorts of things. Trade between Canada and the U.S. was a complicated business, not all of it above-board. And this subsidy was just another deterrent. And Diefenbaker campaigned on an anti-American platform in order to pander to the guys in the wheat-growing provinces.

But this anti-American campaign did not go unnoticed in the U.S. At the time, the U.S. Senate was debating a buy-American policy for armaments. What they wanted to do was to make sure that American armaments manufacturers could persist and flourish. So they voted through a bill which insisted that American sources for armaments be used exclusively, period, with an exemption for Canada, because there was supposed to be a reciprocal trade agreement

[Tape 5, Side B].

Beginning of Session 4, afternoon of 6th August 2005.

HAIGH: You had outlined the political situation, and you were about to tell us how it influenced your career.

KAHAN: Well, it influenced it a lot. When I got back to Canada in 1960, of course I expected I would be teaching students. What I had not expected was the state of the job market. In effect, the job market for the kind of people that I would train had almost dried up. Only the military would justify building the kinds of computers that were suited for the scientific and engineering computations that I was interested in and wanted to teach people about. Naturally, I have to teach students today what they will meet—or what I expect they will meet—when they get out the end of the pipeline. We have a pipeline that is at least seven years long, sometimes more. I use, today, computers with the capabilities that will be commonplace some seven years hence. And those computers cannot be justified by their commercial market today. They are justified by the needs, typically, of the defense establishment—which is the largest, by far, of the consumers of advanced computation. And that defense market had dried up in Canada. In consequence, there really wasn't a great deal of demand within Canada for the kinds of people that I would be producing. The demand existed in the United States. There wasn't a great deal of demand for the kinds of computers that I would want to have to teach my students. The demand for those computers was in the United States.

Well, it took a while for that to sink in, because when I came back I still had memories of Canada as when I left. And Canada had changed over those two years in this very important respect. So I was producing students to satisfy, preponderantly, the American demand. I have to say that that was a discouraging thought. It was one of the factors that weighed on my mind when I decided to accept the invitation to come to Berkeley, which was put to me in 1968. But in 1960 when I came to Canada, I didn't know how things were. I knew what damage Diefenbaker had done in the obvious sense of the disappearance of these little shops and the flight of talent to the U.S. I just didn't realize how nearly mortal a wound this was to Canada's independent development of technology. Silicon Valley could have been along Highway 401; that's how vibrant and interesting the activity was until Diefenbaker was elected. Two months later it was

---

with Canada. But then Diefenbaker got on his high horse, and the Senate got offended and they struck out the exemption for Canada. Which meant that the Avro Arrow was not going to be sold to the U.S., if you believed this.

Now, look: you have to understand that if the Americans weren't going to buy the Avro Arrow, then they were going to fly inferior aircraft. That wasn't going to be tolerable, so a reasonable person would understand that they could strike out this exemption today and put it back in next week. But that was not the climate at the time. Diefenbaker was elected. The liberals had been in power for a long time, and there were lots of things that people could be angry about. Diefenbaker was elected, if memory serves me properly, in 1957 and, within two weeks of his ascendancy as Prime Minister, all three Avro Arrows had been cut up for scrap. This was a symbol of the previous regime, and Diefenbaker wanted none of it. He felt that the farmers and the loggers and, maybe, the miners were the salt of the earth—and everybody else was, at best, to be distrusted, if not altogether reprehensible. Within four weeks after that, all these little shops were closed. All their people had been spirited away to the United States. I think the airframe went to North American Aviation. I think the engines went to Pratt &Whitney, or someone like that. The electronics all went Hughes. Americans came up in droves as vultures to pick the bones clean of all that talent, and it disappeared within a month. I saw it happening and I knew it was appalling. What I hadn't fully thought through, I discovered on my return to Canada. And if that is the end of the tape, I'll tell you what I discovered.

gone. I think my story is one that others may want to hear just to reflect to contemplate on the consequences of various kinds of political and economic folly.

HAIGH: So returning to the same department that now as a faculty member other than a graduate student, was that a difficult transition or did you just fit right back into things?

KAHAN: I didn't notice any bumps. They knew me; I knew them; the fact that I was now an assistant professor didn't seem to weigh on anybody's mind. I don't think that anyone thought it would be different. There were some things that happened, but not along the lines of the question you put.

First of all, I wasn't returning to the Department of Mathematics; I was returning to a joint appointment in mathematics, and in a new Computer Science Department, and there was a certain amount of excitement in that. I didn't realize then that numerical computation would soon become no more than a sliver under the fingernails of computer scientists. I thought computation was my life, and that was what computing was all about! Oh boy, was I wrong. But I had to roll with that punch. The mathematics department was hospitable and I particularly admired and respected the new chairman who was appointed, if not already when I got there, then shortly after I left. That was Daniel DeLury. He was a man whom I had taken a course as a student in experimental design. He was a statistician, interested mainly in agricultural statistics, and I got along very well with him. I taught math courses, typically numerical analysis courses.

Something horrible had happened. I was going to work with Jimmy Chung, who was a statistician, very busy with numerical computation. But he died in the summer of 1960 before I got back to Toronto. He always had some lung problems and finally I guess they overtook him. He was a young fellow and not a hell of a lot older than I was, maybe ten years. The other person with whom I was going to work was Boris Davison. Boris Davison, originally Russian, but had lived I think in England and certainly in Canada, and was working for the Canadian Atomic Energy Commission during the war years, now had some kind of position at Toronto. Don't remember exactly what it was, but we three were going to be numerical analysis at Toronto. And Boris Davison died in October or November in 1960 of a heart attack. In fact, Gotlieb and I went to his house to find out why he hadn't turned up, and Gotlieb went upstairs and found him in the bathroom; he had died of a heart attack there. And so I was the only numerical analyst. Uh-oh!

Well, of course that meant I was teaching the courses and planning the syllabus but, the fact was, that there was a lot of work to do. At the same time, I was active in the activities of the computing center at Toronto, and became active as a contributor to SHARE.

HAIGH: Had the 7090 been installed by then?

KAHAN: Yes. The 7090 must have been installed in the spring of 1960 or somewhere around there. Because the 7090 was already there and, although it was running emulator programs as I mentioned to emulate the 650, it really was a horse of a different color—a much better machine to use. To begin with, it had 32,000 words of core memory, its cycle time was two microseconds, which was quite an advance on what we'd had.

The way the machine was used was that you punched a deck of cards, and it accepted Fortran. At that time they graduated to was what called Fortran II. You punched a deck of cards with your Fortran program. If you were thoughtful, you would serialize the cards just in case you dropped them. Then you would bring them to an operator, who would pass the cards through a reader to pass the card images onto a tape on a machine called 1401. After enough of these batches of card images that accumulated on the tape, it would be transferred to an operator running the 7090.

The 7090 would read the tape; it would read your card images, do what was needed—typically it was a Fortran compilation, that's what most jobs were. Typically it would reject your job for some compile time error. I kept statistics and I found that, for most people, they would run eight jobs in order to get one that would actually do some computation, was what they had intended. One out of eight—that is dismal productivity.

HAIGH: How long did you have to wait to get your output back from the person--?

KAHAN: I was about to say that. If you got two runs a day, you were doing pretty well. Your output would be either a card deck if you asked to have it punched, or a pile of paper—it was fan-form pages, but they were huge sheets. I must say that the most common complaint I would have to deal with, because I was still advising people about computing as they were still coming to me. The most common complaint was, "Where is my output? In all this printed stuff, these piles of paper—where is mine?" That mode persisted for all of the eight years that I was at Toronto—for the 7090, the 7094, and then when we got a 360 MOD 65, which was, I think, in 1967 or '68. They were still doing it in batch mode. That is pretty much how everybody did it. There were some schemes where people were doing time-sharing. They had terminals, and they would use computers in a time-sharing mode. And the 360 MOD 65 was not designed for that, but the 360 MOD 67, which was designed to specifications spelled out, in large measure, at the University of Michigan in Ann Arbor. That was the time-sharing machine, and we didn't have one at Toronto. Even when I got to Berkeley we didn't have time-sharing, although they were working on a time-sharing system.

HAIGH: Yes, with the SDS machines.

KAHAN: No, it was really CDC 6600, a pair of them. I didn't use the SDS machine, and I don't think it was around when I was there; if it was around, it was in somebody's basement.

HAIGH: It may have left just before; I know the work on time sharing for that was commercialized with the Comshare and Tymshare companies.

KAHAN: Well, that is something I wouldn't have had anything to do with. There had been previous machines at Berkeley, and the 7094 still existed in the basement and only Derrick Lehmer would use it. Yes, there had been other machines; they just hadn't been significant anymore. For one thing, the CDC 6400 outraced all of them, and that was the machine with which I was associated. But we are getting ahead of ourselves.

Well, the first thing I had to worry about was not that I ran eight jobs to get one through. What I had to worry about was that I couldn't use Fortran to teach some of the computer science stuff, which would involve graphs, other combinatorial objects. That is an arena in which I had no particular expertise, but it doesn't matter. It was something that I felt had to be taught. So I took it upon myself to modify the compiler. I was just an assistant professor, and you have to understand that I was below everybody else's radar screen, as far as I knew. So I would just go and change things. I had some fellows that were running things at the computing center; as an assistant professor, they thought I had the authority to do things. I didn't have the authority—I didn't ask for the authority, I just did it. But I was very careful. I treated the people who ran their jobs at the computing center as precious guinea pigs. I may have been experimenting on them, but I didn't want to kill them.

So it was my habit to retain a copy of the job tapes from the 1401—both the inputs and the outputs. If I wanted to make a change, I would figure out first, by experimenting in the wee hours of the morning, which jobs had used whatever it was that I was going to change. I could do that because somebody had gotten it into their heads that, someday, we'll pay royalties for the library sub-programs that we use. Therefore the accounting record had a list of the library sub-programs you used. Did you use the compiler, did you use log expansion and so on. These were all part of the accounting record. So I could go through the job tapes and find out who used whatever I was going to change. Of course, almost everybody used the compiler. But then I would check to make sure that if I changed the compiler, the results that they had gotten from the compiler had not been changed by my modifications.

HAIGH: So what did students need that wasn't in the standard Fortran? Was it recursion, better data structures?

KAHAN: No, I couldn't get recursion in there; that would have been a major change of everything. What I could change was things like: what kinds of expressions can you use for indices. Initially, in Fortran, the end expressions had to be in the form of integer-constant multiplied by integer-variable plus integer-constant. Nothing else was allowed; you couldn't have double subscripting. You had to write a separate statement. I said to hell with that and fixed it. Now you could have arbitrary integer expressions pretty narrowly as subscripted expressions with a little tweak in the compiler.

I don't remember exactly what I had to do, but it couldn't have been very much. A compiler program is enormous, so you have to find out where it is and make a little tweak and make sure it doesn't disturb anything else. I can't remember all the changes I made in the compiler, but that won't matter very much because they didn't last for long, and I'll get to that.

Also, I had found that programs I wished to write in order to perform computations as I thought they should be performed would frequently fail on tests that I ran. I discovered, at that time, that the problems resided mainly in IBM's functions subroutine library. So I would change them. Log has to be rewritten; square root has to be rewritten. All right—so they are assembly language. I would rewrite them, turn up with the deck to the people who are in charge of running things at the center and say, "Here, substitute this deck for that one." But remember, I tested these first on the guinea pigs, that is on a copy of their job tape.

HAIGH: Did those functions come with the Fortran compiler?

KAHAN: Yes. If you included a reference to one of the library functions, then its object code version would get loaded with your program, but I can't remember just where it was found to be loaded. You didn't include it in your deck; you just referenced it and it came from somewhere, and it may have been that it came from a library tape. I just don't remember right now how it came. It may have been that it came from a 1401; I just don't know. It's lost to my memory. But I knew that the librarian had a deck for each one of these functions. If I replaced one we were replacing a deck. So that a deck, some card images would get processed and get put somewhere that has, unfortunately, evaporated from my memory.

HAIGH: So the functions came from IBM?

KAHAN: Yes, initially of course. They really weren't all that great. They were slow, inaccurate, and anomalous. Somebody had written them up. I think Cody has described a little bit of the style in which people did it. They would go to some book and find some formula and they figured, if they wrote that formula in Fortran, they would get what they wanted. And of course they didn't.

I remember rewriting the square root program. This is interesting enough to go into in a little bit of detail. I had access to a room that before I went to Cambridge, had held the Meccano implementation of the Vannevar Bush differential analyzer. But that room was now empty, and was going to be converted into some other type of lab. But it was empty for the time being, and I had access to a desk in one corner. I spread a flow chart out on the floor, but not the sort of flow chart that you may be thinking of.

What I was interested in was of all possible programs for computing a square root, which would be fastest? I knew that I had some implementations of square root that, therefore, a program can be arbitrarily long. I knew that some instructions were simply not going to make progress towards the square root, so I wouldn't worry about them. But I would start from the initial instruction, which loads and stores the return address and so on. I would have these branches, and the branches were if you execute this instruction what function have you computed so far? Now, what other options would you like to try? Every now and then when I go down some branch, I would get to a place where the function I have computed here is the same as the function I have computed down a different branch, so I'd stick a piece of ribbon from here to there. Then I don't have to follow that little branch anymore. Thank goodness that pruned the tree sufficiently that, although it covered the whole floor, I could actually get it on the floor. I ended up with what I was convinced was the fastest possible way to compute a square root on the IBM 7090 and get it correct to within several zeros, and then one-something in the last place.

Then it had to be tested, so I tested it on all floating-point operands between one and four—or maybe it was one half and 2, I don't remember right now. Because they all have the same bit patterns in a binary machine. So expect for the extreme cases, where you could run into over or underflow problems—but you wouldn't for square root—there is just no way to get over or underflow when you are computing a square root in single precision. Aside from those end cases, which of course got tested, what I did was run on all two-to-the-27 different arguments between one half and two. It took a while, but I was an assistant professor and I had the authority to say if no one is running on this machine, run this. I found that there was some handful of numbers, couple of dozen or three dozen numbers for which square root was not correctly rounded. If it wasn't correctly rounded, it was out by .50000-something in the last place and that was it! I thought it isn't even worth fixing them. It meant, however, that I could prove that square root was monotonic.

Funny thing happened. Some time later I saw Hirondo Kuki's code for the square root and he had used an extremely similar scheme. The scheme was bizarre, because the first thing it did with the floating-point number was a fixed point-shift-add-subtract, and then, in my case a logical *or* and, in Hirondo's case, a logical *and*. So the only material difference between our two codes was that I used *or*, which took two cycles, and he used *and*, which took three. That was it. It was amazing.

In the meantime, my code was getting copied. I'm told they copied at MIT onto a DEC machine, was it the PDP 6, or something like that. They called it Kahan's Magic Square Root because this first little thing—the shift, the add or subtract, depending on how you did it, and the logical *or*—that gave you three significant bits of a square root. Then all you had to do was refine those three bits by a few Newton iterations, and you had it. Of course there were other ways to get first approximations, but they took longer, even though they gave you better approximations they took a lot longer. So I was in clover. I had proved that it was the fastest way to do it on the 7090. Of course it got submitted to SHARE.

HAIGH: So that was your first routine that you submitted?

KAHAN: I don't know if it was the first, but it was among the earliest. Later I adapted the same algorithm to the CDC 6400, speeding up its square root. I adapted that to cube roots, double-precision square root, and there is a paper I can't find-- When we were deliberating, back in the early '80s, like '81 or so, should square root be one of the functions that is standardized for the IEEE Standard. And there were people that said how can you standardize it you can't compute directly around it? Not realizing that, in fact, you could. I wrote a little paper showing you exactly how you could do it. It tossed you one extra divide over the other algorithms that I had already used. That was it—you could do it in software so, therefore, if you didn't want to build it into the hardware, that was okay. Do it in software.

By 1980, IBM had also found ways to compute square roots rounded correctly on their equipment, even though it was hexadecimal. I think they called it a Tuckerman Test, after Bryant Tuckerman who was really a number theorist but who was working with Fred Gustafson and others on their math library. I remember this mainly because I remember spreading the thing out of the floor and having to get a box of thumbtacks and some ribbon in a computer room. Why would someone want some ribbon, like little ribbon used for sewing? Of course they didn't have it, so I had to get some of my mother's and tack it in.

It was fun to live a life where you could enjoy that sort of luxury of finding the best possible program, and so on. Well, but also if you look through here you will see that there are other programs. I modified log, exponential, y to the x, I think I did sine and cos. And I was teaching courses. Teaching numerical analysis courses. The first numerical analysis courses used the IBM 650, that still was hanging on. Very soon after that, the courses moved to IBM 7090.

One of the courses I taught caused somewhat of a stir. It was a numerical analysis class taught to first-year students. It was an attempt to revive a course that had been taught in the physics department by a professor by the name of [John] Satterly. Satterly had taught us in physics a little bit of calculus, and a little bit about statistical computations and so on, because the physics department didn't trust the math department. Well, Satterly's course was a bit of a joke, but it did teach people a little bit about computing with slide rules and things like that, nothing outstanding. Here we were thinking that first year students ought to have something about computing because they are going to go on with labs and stuff like that. So I taught them a little bit of this. It was my exam that caught people's attention. You get eight marks for every correct decimal digit and you lose eight marks for every incorrect decimal digit. You don't have to get as many decimal digits as one could conceivably get, but don't claim more accuracy than you believe you deserve—it's done with a slide rule. And, oh, if the students didn't have a slide rule, that was okay; I had a collection and I could make sure that anyone who came to the exam and said, "I forgot my slide rule," I would say, "Here, try this one." That turned out disastrously because, of course, the students were too petrified to claim any reasonable number of decimal digits. Although they need to check one's computation in diverse ways. If you understand anything about the process, you could easily get one digit. If you understand enough to use a good way of doing the computation, you could get two and know it. And if you actually knew a little bit more of error analysis and some of the things I taught you could get three—which was pretty much the limit for a slide rule.

There were people who ended up owing me marks, and I can recall that afterwards…I think it was called G. de B.'s Compromise…just add 50 points to everybody's grade. Nonetheless, there are still students who still remember that, perhaps, because they are scarred for life. I think one of them now is a fairly prominent professor in New Mexico, or is it Arizona? But anyway, I remember some of the students. There were some who did extremely well and she was one of them. I don't think I should name her because I don't want to embarrass her at all. She did well. There were three students who did superbly well, and two of them—I have followed their careers. The third, somehow, got lost into chemistry and disappeared from the scene. I was certainly able to identify the good ones or the very, very good ones. Or maybe what happened that that was such a traumatic experience that those ones who were good but not lucky enough decided to leave. We don't know and it's very hard to tell, isn't it?

Well, I taught about lots of things and had some graduate students. But gradually the involvement with SHARE started to loom larger and larger.

HAIGH: Let's talk about SHARE, then. Pretty much every owner of 709/7090 kind of family installation was a member, so when you came back and the machine was there, were there also large volumes of SHARE materials and manuals and so on for you to review?

KAHAN: Yes, well I don't know large they were but Beatrice Worsley was there and she was the one who was responsible for managing that stuff. She was the official liaison with SHARE, as I recall. My interactions with SHARE were principally the contribution of library sub-programs, which we deemed sufficient quality to be worth distributing. We could not distribute the modifications to the compiler. SHARE wouldn't accept that, they didn't have a way of accepting it so those modifications remained homegrown and stayed there.

But in any event, we were destined very soon to be annoyed because IBM knew, as we did not, that IBM was planning to bring out the IBM 360 series. Its most distinguishing characteristic was that it was a multi-register machine, whereas the IBM 7090 and 7094 and so on had inherited from its predecessors what might have been called a von Neumann architecture. Namely, you've got an accumulator called the *AC*, and it has an extension that is called the *MQ*, and the MQ is where you find the right-hand part of the product if you do a multiply of two single precision words, one in the AC and one in the MQ. You multiply the both of them and the AC/MQ is replaced by a double-width product. And if you want to do a divide—at least in fixed-point—you put your numerator into the AC and MQ, and then you do a divide, and you end up with the remainder on the AC in the quotient on the MQ. And that is the way it worked in fixed-point, and that's the way it ended up working in floating-point except that, if you did a divide, the right-hand MQ word was not regarded as an extension of the AC. But if you did a multiply of two single-precision floating-point numbers, you would end up with a pair of floating-point numbers—leading digits in the AC and the trailing digits in the MQ. They were mimicking the software that they had inherited from the 701 and the 704. The 704 didn't start out with floating-point—at least I don't think it did. But it got floating-point hardware to mimic the software. Well, mostly mimic, there were a few other things.

What we didn't know was that IBM was planning for movement to a multi-register machine with an altogether different architecture. That meant that some of the semantics of arithmetic expression handling would not have been so convenient on the new machines. Also they were planning to switch from binary to hexadecimal. There was a paper and the name of the author was Sweeney, in which they had thought that, for the sake of their cheaper machines—the ones with the narrower internal buses—they could reduce the incidence of shifting if they used hexadecimal arithmetic. [D. W. Sweeney, An analysis of floating-point addition, IBM Systems Journal vol. 4, Issue 1, 31-42 (1965)].

Cody has a couple of words to say about hexadecimal arithmetic, and why he didn't like it. He said they used to lose a digit, but that was not intrinsic in hexadecimal. The problem with hexadecimal (as Cody knows full well, it just didn't get out in your interview), is what Cody called *wobbling precision*, he coined that term. What that means is that, in hexadecimal arithmetic, the density of numbers jumps by a factor of 16 as your numbers cross a power of 16. So as you go from numbers a little bit less than one up to numbers a little bit bigger than one, the density of numbers available to you drops by a factor of 16. That is because you are carrying a certain fixed number of significant digits. In decimal, the same thing could be understood. If you are carrying three decimal digit numbers just less than one—0.997, 0.998, 0.999, and then 1.00. Their differences are all ten to the –3. But the numbers that are just barely bigger than one to three significant digits, they are 1.01, 1.02, and 1.03, their successive differences are ten to the –2. The difference in density jumps by a factor of ten, the radix, when you pass a power of the radix. That was the headache involved with hexadecimal arithmetic. It meant that the density of numbers varied more abruptly, more violently. Therefore, when you performed a computation—which later would somehow get shifted so that numbers originally just bigger than one and a little bit less than one would produce results that might be somewhere between three and four—the trouble was that the rounding error inherited would be 16 times as bad for some numbers as others that were neighbors.

HAIGH: So presumably in binary you have the same thing, but it would be a factor of two?

KAHAN: Just a factor of two, that's right. Exactly. A second problem with hexadecimal is that the trade-off in range and precision is somewhat more brutal. On the binary machines, the numbers ran over a range of ten to the ±38. On hexadecimal machines, the range was approximately ten to the ±76, which seems like an improvement—but it was secured at the cost of lower worst-case precision. It turns out that's intrinsic in hexadecimal arithmetic as it was understood at that time. Without somewhat bizarre encoding, that's intrinsic. If you use hexadecimal instead of binary, you are destined for a given range to lose two significant bits of precision, in worst case.

Sweeney's experiments had not revealed the full horror of the situation. His experiments had revealed that, if you use hexadecimal, you will reduce the incidence of normalization shifts. On the cheaper and smaller machines in the 360 line, the normalization shifts would be a bit of a nuisance because their bus was rather narrow. The use of hexadecimal penalized the machines at the higher end of the 360 line, because those machines had to have binary shifts anyway—everyone did. So having to shift in clumps of four bit shifts was actually a nuisance for the 360 MOD 65 and up, the faster machines. But of course, IBM understood—quite rightly—that the bulk of the machines they were going to sell were going to be the lower end machines, at least initially. They were optimizing the design for the lower end machines and, therefore, pessimizing it as far as we were concerned, for scientific and engineering computation. Hexadecimal was going to be a real drag, but we didn't know that.

The way IBM prepared for the transition was to introduce something called *IBSYS Version 13*, which was a new version of their IBM operating system for the 7090/7094.

HAIGH: What was an "operating system" understood to consist of in the days of the 709X machines? I believe there was a monitor and some kind of job control system. Is that right?

KAHAN: Yes, but remember what I said the process was: you batched your jobs; they went up on tapes produced by the 1401; it was mounted on the 7090 and, similarly, on the 7094 that read these tape images. The operator would watch, didn’t have a hell of a lot to do. If a job crashed and hung up the machine, the operator would have to reboot, which took only a moment or two, and continue to the end of the crash job and begin the new one.

HAIGH: So would the monitor be loaded permanently into memory?

KAHAN: They had somewhere to load it from, whether it was a disk or a tape dedicated to that purpose I can’t remember. Some of the monitor was permanent in memory, of course, but the bulk of the monitor was definitely not. We had almost all 32,000 words to play with.

HAIGH: At that point, would the operating system have resident modules loaded to handle input and output or was all that--

KAHAN: Yes. They had channels and those channels were little computers in their own right. There was an interface; there were instructions whereby you could send instructions to a channel to take a certain amount of memory and copy it to some output device and so on. I actually didn’t play with that much. I just took the channels for granted.

HAIGH: So those would be done in hardware rather than in software.

KAHAN: The channels were little computers in their own right—yes, hardware. Absolutely. Nonetheless, there was trapping because, when the channel was finished and wanted to tell the mainframe computer “give me some more to do,” it would trap, and that would interrupt the mainframe. Then it would, perhaps, send another buffer-low to a buffer—tell a channel, take this and then, while that was going, it would then go back to whatever it was doing before. Why are you asking, I’m curious?

HAIGH: I’m interested in what early operating systems did, where the idea of an “operating system” came from and how it changed over time. Today, you take it for granted that an operating system will be completely mediating between peripherals and programs, so you just give it a file name and it works out what disk it’s on and all the blocks and reads and writes things. And we take for granted that an operating system is doing memory management and those kinds of things. I know that in the days of SHARE, back in the ’50s, people first started talking about an operating system; it seems like what it was originally conceived as being, was to some extent a system that would do what the human operator had been doing in terms of providing some monitor capabilities in terms of maybe helping to batch jobs instead of having had that done completely manually. It seems like people would think that an assembler or a linker and a loader as being part of the operating system, too.

KAHAN: Well, no. Linker and loader, yes. The assembler was a separate program, whose task was to read your text and produce a new text in a form the machine liked to eat better. Loader—yes, that was part of the operating system because, as I said, if you invoked a library program you didn’t have a copy of it with your program. What you had was something that got linked in from elsewhere, and there were addresses that had to be filled in. That was the loader’s job. And then your computer was running under the supervision, so to speak, of a monitor; if your program crashed, then there was someplace you had to go or the machine just simply stopped. Of course there were programs that got into endless loops and they just wouldn’t stop. It was initially the operator’s job to look for such things, but-- oh, you’re going to love this!

There was a clock associated with the printer, and this clock sent out a signal at regular times; I don’t remember the rate, but it was reasonably frequent. This signal was essentially a signal that said, “The printer is ready for your next stuff to get printed.” It would come to the printer on a channel, and that would interrupt the processor. What I did was to put into that interrupt-handling program something that would look to see whether the instruction upon which the program had been working when the trap occurred, was a divide. And if it was, was it dividing by zero. Because if it was dividing by zero, and you hadn't masked off the trap for divide by zero, then you get a zero quotient, which was dangerous. So I enabled a trap, turned the trap on, and if it was dividing by zero and, therefore, stuck, then the operating system would initially terminate the job; later it called *uncle*, and gave you a message that said you had hung up on a divide by zero.

Previous to that it would have been the operator's job to do this and notice that somehow the computer had stopped. But now the operator didn't have to and something like that, somehow, found its way everywhere. I don't think we were the ones who necessarily were the only ones doing this, or even had originated it. It was the most bizarre interaction from the printer clock to the mainframe to get something like this, but now you'll understand the desperation that we faced. We had to do something by hook or by crook.

[Tape 6, Side A]

HAIGH: So just looking at one of these documents that you provided, it seems that the way the IBSYS system seemed to include a monitor also called IBSYS, a loader and library called IBJOB, and a Fortran compiler called IBFTC.

KAHAN: Yes.

HAIGH: So, collectively, those things would form what was thought of as being the IBSYS operating system.

KAHAN: The compiler came with the operating system, without a doubt, but I don't know if you could regard it as part of the operating system, because it was an application that took card images and produced other kinds of images.

HAIGH: That's true. I'm just looking at the way it is listed there.

KAHAN: Well, these are just the manuals. The IBSYS monitor was a program, part of which was resident. But they got the rest of it from somewhere—it may have been actually off a drum, I don't remember—or off a resident tape. But I know that they couldn't have occupied very much of the memory, because if they had, we would have had very little to play with.

HAIGH: So you think, mostly, there would be just enough things resident to deal with all the interrupts?

KAHAN: Yes, absolutely. There has to be enough for that, the interrupt connectors and so on. Once the interrupts would occur, some of these had to be handled on the fly, like the floating-point related interrupts, and others had to be handled by actually swapping out your job. Usually they were going to kill your job, anyway. What you might have liked was a core dump, but it wasn't a very easy thing to understand if you got it. For one thing it was an octal and mercifully, I have forgotten much of that. I am grateful for the ability to forget people and things that I dislike. So I had modified the compiler, and I was modifying the library, and then IBM replaced IBSYS Version 12. The Fortran II compiler I had modified, and then I had modified the Fortran IV compiler. But when we got to IBSYS Version 13, I couldn't modify the compiler anymore. For one thing, it was just too horribly obscure; for another, I think I was forbidden at that point to tinker with the compiler.

HAIGH: Why?

KAHAN: I can't remember. It may have been an IBM system engineer, but I don't remember who forbid me. Thing is, we now had a system which had been standardized enough by IBM in which they had a sufficiently jealous interest that they didn't want us to tinker with it. Previously, if we tinkered with something, it could have been an improvement from SHARE, or it was just a local thing and you were on your own hook. But now they really wanted to have this uniformity. And what was happening was that the semantics of the arithmetic expression evaluation was changed in subtle ways that most people didn't appreciate in order to match the upcoming semantics that they associated with a multi-register machine.

Let me give you an example of the difference. Previously, if you multiplied two single-precision numbers together, the product would appear in the AC and MQ as a double-precision product. Then you could add it to a double-precision sum, which is what we wanted to do with matrix computations. But on the 360 as initially conceived (this did change), they intended that the product of two single-precision numbers would be construed as a single-precision number, period, to be added to other single-precision numbers; you would have to do a coercion explicitly to turn it into a double-precision number, and then the registers would be used differently. The IBM registers were wide enough for a double-precision number, but single-precision instructions really paid attention only to the leading half. So they just used their single-precision input, and put their output in the leading part of the word. There were a couple of exceptions, but they came into existence later. So they really wanted us to change our ways so we would use the IBM 7094 in a fashion that would more nearly match the architecture of the 360 (which, of course, we didn't know anything about, so we didn't understand their reasoning). But we did understand now that evaluations of expressions were different, and places where we had exploited this for multiply and add—add did the same thing—you add two floating-point single-precision numbers and you got two floating-point numbers and the sum, a leading word and a trailing word. We had exploited this heavily, and now the new compiler was hostile to that sort of stuff. We didn't know why. The new compiler already allowed more elaborate indexing than we'd used before, and so I was still able to use the compiler for the graph theoretic stuff and some combinatorial things. It was the floating-point that got damaged the most.

Another thing that happened—it actually happened going from Fortran II to Fortran IV. Fortran had allowed two modalities for passing arguments. The first modality would pass by value, which meant it put the value of the argument in the AC, called the function and expected to find the result in the AC. Functions that did this had a suffix, *F*, on their names. In Fortran IV, they stopped using that and attempted to arrange that every function would pass its argument by name. In other words, you would pass the address of the argument and, therefore, to retrieve the argument, you would have to first load the address and then go find the argument. Because memory speed was not particularly slow compared to processor speed, this didn't seem like much of a burden. But it meant that some of the kludges that we had used would no longer work.

For example, I had FRNDF—what did that do? Normally on the 7090 and 7094, if you performed a floating-point operation, the leading word that you would get is the result of truncation—you just throw all the bits past the 27th away. That is an unfortunate thing to do sometimes, and so I had an FRNDF (or it may have just been FRNF, I don't remember now). That was something that I produced as a library function and passed it on to SHARE, that would say "no" for at least for adds and multiplies and, to a lesser extent, for divides. If you said FRNF of an expression, the last arithmetic operation in the expression would get rounded instead of truncated. How did that work? FRNF was a function that you had to call because it was an *F*-ending function; you would call it on the operand and the AC and MQ. You would execute a TSX instruction, which transferred to program and then it would return, and you would continue execution. All it did was change the TSX to floating-point op code from the machine, an FRN, which took the second word's leading bit and used it to round the leading word's bit. That's all. What I had was a function which, when first called would overwrite itself—that's all it did. It overwrote itself and then was an execution. Next time you passed that way, you didn't call anything, you just executed the round instruction, which had overwritten the transfer. This is a kludge, and it worked. Couldn't do that anymore.

I could go on and on, there are a lot of things that had to changed when we moved to Fortran IV, and a lot more things that had to be changed when we moved to IBSYS Version 13 and the new Fortran compiler. As we had a discussion a little while ago, you tried to improve technology only to discover that, if you can't do it instantly you may find that you arrived at a destination that isn't where you began, things aren't the way they were, and you can go nuts trying to figure out now how to re-adapt your work for the new environment.

HAIGH: It seems from the report that you wrote that you were able to reproduce the system in the new version of IBSYS, is that correct?

KAHAN: Up to a point, yes. FRNF now ended up as a subroutine call. You really did have to do something, it now cost you something. There was the DLAD-DLAP package and so on; all these things had to be changed to adapt to the new environment and it was a bloody nuisance.

HAIGH: In the document in which you wrote things up, you said the floating point tracking capabilities and the treatment of underflow and overflow in the monitor were very important. In fact, you say with reference to one of the features, "The author is under the impression that the new FPTRP's treatment of improper divisions is more widely appreciated than all his other works."

KAHAN: That's correct. That's why I told you about this—it was originally a printer clock, but with a newer machine, you had a divide-by-zero trap. And you ended up in the floating-point trap and you could detect that it was a divide-by-zero, and you didn't have to wait for the printer clock to tell you. That was marvelous, it isn't as if any change is bad. That's why I regret that I can't find this source code for the floating-point trap: I know it's somewhere, I just don't know where.

HAIGH: This is something that you achieved while rewriting the standard monitor program?

KAHAN: Well, the standard program loaded as part of the math library; it wasn't the monitor, it was part of the math library. But it looked as if it was part of the monitor, and when you would trap, people would think you were trapping into the operating system, but you weren't. All the monitor had done was to aim the trap to the floating-trap handler. That was the only part that I had to rewrite. In effect, it was no more than rewriting a log function or exponential function or whatever, although you would have thought it was part of the monitor. There was some connection, because there were flags; I had to put the flags somewhere where the monitor knew about them in order to get messages about unrequited exceptions. So I had to rewrite the monitor a little bit. Now what that amounts to, unfortunately, is rewriting the accounting system. The monitor keeps track of how long you are on the machine and then, when your time quantum is exhausted, there has to be a trap (unless you've already exited). There's a little bit of monitor that checks on that stuff.

We had some very clever people who used to extend their computing quantum by diddling the monitor into thinking that they had more time. That was one of the little things we had to fix. So I wrote a piece of code that had undocumented instructions in it. The time allocation was obscured by the use of these instructions, so that people could not figure out exactly how we computed how much time they were there—they couldn't figure out exactly where we put this, so they couldn't do that anymore. It isn't as if we had a bunch of thieves around, but there were some people, as the English would say, too clever by half, and so this was part of what I did to defeat them and, at the same time implement the IFKICKOFF and UNCLE capability that is written up there.

So yes, I did have to rewrite part of the monitor; yes, I did have to rewrite part of the accounting system, because I wanted the accounting system to include a message that you had an unrequited exception, an overflow, a divide by zero, whatever it was, and tell them where it happened. And telling them where it happened was meaningful because, although there were program overlays, programs weren't normally translocated, so you knew what the octal addresses were the parts of your program that came out as part of the compiler's listing if you wanted it. Of course you'd end up with a lot of paper. So if I said you had an unrequited divide at such-and-such an octal location, you could look at your program and see a divide there, and then you'd look at the compiler listing to find out which statement in the compiler corresponded to this part of the assembly—or pseudo-assembly—line listing, and you could find out which instruction your program had caused the divide-by-zero, presumably. Then you could fix it, and that's why I got thanked.

HAIGH: So you say that, except for a few programmers you describe as a lunatic fringe, everyone was much happier with this. So how did you get the word out to the users—did you have to write up some documentation?

KAHAN: Initially, of course, it was documented—but who reads documentation? Of course it was all documented; of course the programmer's reference manual was updated; of course we had little notices here and there about stuff—but who paid attention to that? No, they discovered it because it would happen.

HAIGH: They would get these extra messages in their error?

KAHAN: Right, that's correct. Not everybody wanted to thank me for those messages because it sort of spoiled the accounting record a little bit. Remember, my guinea pigs were precious and I couldn't afford to kill any. So I had to make sure that whatever it was they were doing, if it was something with any claim to legitimacy, I wasn't going to spoil it. That's why, in the wee hours of the morning, I would come in and rerun those tapes and other things just to see what happened. I tried to be responsible and, I think, that's why I got away from it. Clearly I wasn't fermenting large numbers of complaints, or else somebody would have gotten onto my case.

HAIGH: This was an opportunity that you had to alter parts of the operating system, the compiler, and the other aspects of the system in order to give programmers meaningful feedback about what had happened with the floating-point errors at run-time. It seems that this may well have been an important event in your career, inasmuch as many of the manifestos and attacks you have written on systems that are around today are in some senses faulting them for failing to achieve this same level of integration.

KAHAN: Absolutely. We are getting close, and almost to the point of the big tragedy of that era. I was writing programs. Some of these programs are very devious, like the JCPM program that was a zero finder. It was rather unusual. It was extremely fast and robust for its time. Now the best and most robust equation solver for solving one equation and one unknown is built into the Hewlett-Packard handheld programmable calculators, like the 28C, the 19C, the 45G, and so on. They have very good equation solvers, which are the descendants of this program. You can find some write-ups on my Web pages about real roots of equations and some lecture notes there…nowhere near complete, but, anyway.

Among the programs that were developed were programs written by students. I had come to understand what many academics seem not to understand, which is that a professor's principal product is not papers or programs, but people. Of course we don't make them from scratch; this is a value-added process. A certain number of programs were set aside to be used as vehicles for training students. I want to say something about a couple of those. These were the differential equation solvers. One of the students, Neal F. Stewart, he wrote a program to solve initial value problems using a Runge-Kutta technique. Another student, Gerry Gable, wrote a program using a predictor corrector. By this time, Tom Hull had entered the picture.

Let me back up a little bit. It was clear that just having one teacher to teach numerical analysis was not the best idea in the world. It was doubly clear that if I was that one person it was certainly not the best idea in the world, because I was considered rather demanding. I didn't want to pass students; I wanted students to do the work. If they did the work and demonstrated that they knew the material, then I was delighted. If they didn't do the work or didn't understand the work then I was indisposed to pass them. In any event, my presentation was somewhat crisper, and might even have been interpreted as unfriendly, because I didn't want to be construed as the students' friend. I was there to present that material and, if possible, to present the material in such a way that my personality would disappear. That was my goal but it didn't work out that way, did it? I couldn't make my personality disappear no matter how hard I tried. Surely that was my intent, to present that material so that it would be the material that would live in the student's memory rather than my idiosyncrasies. But I failed.

HAIGH: Did you stop trying to do that later on?

KAHAN: No, I've never stopped trying to do that. It's the right thing to do, it's just I don't know exactly how to do it. It was realized then that something about me would frighten undergraduates and, gradually, my teaching load shifted to a higher level of graduate courses. We clearly needed somebody else, and we found him at the University of British Columbia. That was Tom Hull. Tom was a numerical analyst from way back. He started off, if I remember rightly, as a meteorologist and had done a lot of numerical computation. I don't know exactly why he was less than fully happy about his position at the University of British Columbia, but he certainly jumped when he got the chance to come to Toronto. I was delighted. Even though he was senior to me in principle I was delighted to have Tom there, and it didn't occur to me that he was my senior; that was not our relationship. (He was a full professor, and I think at that time, if I wasn't an associate professor yet, it couldn't have been long in coming. I don't remember exactly when I got promoted. I don't remember why I got promoted; it just happened). In some ways he was my student, and in some ways he accepted that, in some respects. In other respects I would take cues from him because he was politically much more astute than I was, and was much more able to figure out how to soften the blow. So there were some things I learned from him, not as much as I should have. There were things that he learned from me. We had what I regarded as a pretty good partnership. And he would teach more of the lower-level courses than I would.

He would take on certain graduate students; actually, he brought a graduate student with him from British Columbia, someone who, let's say, could not possibly have made a good first impression. He was a very strange bird, and Tom had not succeeded in getting him to do very much but still felt obliged to sort of drag him back over to Toronto with him. Tom and I managed, one of us moving his left hand and the other one moving his right, so to speak, to get this guy to write a paper in his area—which was, essentially, approximation theory—and it got published. On the strength of that publication, we could recommend him for a job. We sent him off, then, to a job at a nearby university where his knowledge of approximation theory made him attractive to the faculty member who was in charge of hiring. Then it turned out that our guy actually knew more about approximation theory than that faculty member, so his position was pretty secure. So far as I know, he was still there, at least until he retired. That was one of our products. I mentioned that so it would be understood that Tom and I had a pretty happy working relationship, even though it might have seemed very unlikely because people would have thought he was senior. He was a full professor and I was just some junior character. That was the way the outside looked at it and, between us, our relationship was more about a matter of peers. I knew how to analyze numerical situations very quickly as compared with Tom, and Tom understood a bunch of other things better than I did. So we got along just fine.

Neal F. Stewart was Tom's student, in principle. Gerry Gable was my student, in principle. We actually shared. Neal and Gerry wrote their programs to look identical to a user. The user who wanted to solve an initial value problem could choose Neal's Runge-Kutta or Gerry's predictor corrector, and call either in place of the other if he changed his mind. These programs were self-adapting; those chose their step size automatically. They accepted tolerance in a way that is still controversial; I'll come to it. They produced some fairly beautiful solutions. Gerry's was based upon ideas that I had cultivated way, way back in the times of the 650.

I think I mentioned Coxeter to you, and I may have mentioned Schlafli integrals. Coxeter was interested in close packing. How many spheres can you put into a barrel in an *n*-dimensional space? In order to deal with this problem—and it was a very difficult problem, and progress has been made on this problem very, very recently. At least asymptotically, for very large barrels compared to spheres, Coxeter had to compute things called Schlafli integrals. These integrals are unfortunate because, in the course of computing them, you run into rather nasty underflow problems. If you try to get rid of the underflow problems, all you do is get into overflow problems. So back on the 650, if memory serves me rightly, I had figured out that the right thing to do is to compute not the Schlafli integrals, but their ratios; the ratios behaved more tamely. Unfortunately, the ratios satisfied a differential equation, which we call a singular differential equation with a regular solution. That is the differential equation itself has a spot where, if you get to that spot, some things become indeterminate—you end up with zero-over-zero-type problems or even nonzero-over-zero, which can be worse. However, there exist regular solutions that are entirely smooth that go through the singularity without a hiccup. So I found a way to get the differential equation solver that I used at that time to solve the differential equations and to give Coxeter his Schlafli integrals—initially as ratios and then, of course, you just multiply it and you get the Schlafli integrals themselves, until they underflow. Then there is no point because, once they get below the ten to the –99 on an IBM 650….they're gone.

Coxeter was delighted and I conveyed these ideas to Gerry Gable, who wrote a very interesting master's thesis in which he developed a theory of these predictor corrector methods, including the important ability to solve for a regular solution of a singular differential equation if you started it right. This will become a very important thing later in our story.

The differential equation solvers were marvelous. They ran on the 7090/94, they required the least possible effort on the part of the users to use, and they produced extremely reliable solutions, as differential equation solvers go. There is always an element of sin in solving problems in the continuum by discrete methods, because you can always jump over when the turn out to be a terribly violent spike that you didn't notice. Therefore you are in the wrong place. There is always some element of risk. These, as such differential equation solvers go, are extremely reliable. In some ways they were too reliable, which is the controversy I want to mention.

A technicality: when you tell a differential equation solver to solve your differential equation, how do you tell it how accurately you want it to work? Without going into a lot of details, you can ask it to contain its estimate of local error below a certain threshold per step. Which means in every step (it's an inchworm process) you solve the differential equation by taking a little step which, is if it's too big, will be detected as being not all that accurate; you then shrink the step. And then you take another step and so you proceed like an inchworm. I think inchworm is familiar to Englishmen, isn't it?

HAIGH: Yes.

KAHAN: Okay, so you know what an inchworm is like and you can just imagine doing it. If you can take longer steps, that's great. There are various means for checking on your accuracy. You can't be 100% sure, but you can get a rough idea of how bad the error is and, if the error is too big, you shrink the step. The typical criterion used nowadays is to keep the error below a stated requested threshold per step. This is a good thing for differential equations that are stable. If the solutions for the differential equations tend to cleave together—which means they tend to forget their past—then a limit on error per step is a good thing; it means at no time will your error, in fact, be much bigger than the error per step, since previous errors tend to have been forgotten. So it's only the last error that is the major contributor if kept at below your tolerance in error per step.

This is not a good scheme if what you are interested in is calculating planetary orbits or, for that matter, calculating trajectories of ballistic missiles. What you want then is an error per unit step, which sounds terribly similar but it's crucially different. Now what you say is if I take a longer step I allow a bigger error. The error I allow is proportional to the length of the step. That means you have to take the step size into account when you ask about the local error. But it also has a very important reinterpretation, namely that the threshold on error per unit step can be thought of as how much uncertainty you are willing to add to the driver on the right-hand side of your differential equation. The differential equation isn't all that certain to begin with. It's a model of a physical situation. If you are modeling planetary orbits, for example, how do you model the effect of a gas that is floating around in a solar system and the planets bump into the gas and there's a certain frictional drag. How do you take into account that the planets aren't just spherical bodies; they are somewhat lumpy in places? There is a certain uncertainty in the strength of gravity and various other things, and we can estimate what the uncertainty is. But what that means is that when you set the differential equations of motion that we have inherited from Newton, and you modify them with a relativistic term if you must, you don't know everything exactly. So that means that on the right-hand side of your differential equation, the left-hand side says the derivative of my unknown function is this expression.

You have to add to that expression plus a little bit. But I don't know. I've got an idea of how big it cannot be. This thing I don't know can't be bigger than so much. The threshold on error per unit step is just a contribution to that term. I know it's there; I know how big it cannot be…and I'll add a little bit and allow the differential equation solver to muck it up slightly more. That is when you place an error per unit step it is a lot easier to interpret. Otherwise, for orbit calculations or for unstable differential equations or even for some switching circuits where the time for switching matters to you, if you place a bound on error per step it's impossible to figure out how wrong your solution is at the end.

HAIGH: So where did the controversy come in?

KAHAN: The controversy came in that, if you place a bound on error per unit step, you are generally placing a much tighter bound than is worth having if your differential equation is stable, as many differential equations are. It doesn't matter how big your error was back there; you're going to forget it pretty soon. The bigger it is, the longer it will take to forget but you are going to forget it.

What happens then is that our differential equation solvers were, in one sense, easier to use and, in another sense, too conservative. They were solving a differential equation only slightly different from yours, but if you wanted it to solve for solutions to very stable differential equations, we were solving them better than they had to be solved. Nonetheless, they were easy to use. It was easy to figure out by comparison what the consequences were in a backward sense. That is to say, you were solving a differential equation not all that different from yours and if differential equations that are not all that different from yours have solutions that you are happy with, then you have to be happy with what we compute. It's like backward error analysis. Not perfect, not justified all the time. But it was a pleasant thing to have, and Gable would solve for the regular solutions of singular differential equations. Which meant that chemists and others could very often solve differential equations with Gable's program that couldn't be solved by anybody else's.

HAIGH: Were those programs distributed outside Toronto's own installation?

KAHAN: They were distributed as master's theses. And the decks were available to anyone who wanted them. They may have been submitted to SHARE, but I don't know. In any event, they had a lot of contemporaneous differential equation solvers. Most of which came out of places like Livermore or Sandia or Boeing, and so on, where everybody had his own idea of what kind of differential equation solver to use. I remember Fred Krogh, down at Jet Propulsion Lab, had his version of a variable-step predictor/corrector method. I didn't like his nearly as much as I liked Gerry's, but our differential equation solvers did not compete in speed well with the differential equation solvers that used error per step instead of error per unit step when it came to solving stable differential equations. Most people believed, rightly or wrongly, that the differential equations were stable. If you look at Matlab now, you will find that they still use error per step. You still go nuts trying to figure out what the consequences are to your final solution. I think people should have a choice, and it should be possible to make a reasonably enlightened choice. What is the phrase…something "consent" for medical?

HAIGH: *Informed* consent.

KAHAN: *Informed consent*—that is the phrase I am looking for. It should be possible for somebody to choose in an informed way which kind of differential equation solver he wants to use. I'm going to explain in a little while why we haven't got that, because that was one of the horrible things that happened later.

HAIGH: While we are on the subject of students at Toronto, I see you added the name of three of them to the outline notes?

KAHAN: That's right.

HAIGH: What do you have to say about those--

KAHAN: Dev Chowdery had come to us from India, via Britain. Dev Chowdery was the son of a wealthy family in India. One day he took a peek at the bride who had been chosen for him and decided that he would rather not, and skipped out of town and came to Britain. He got a job there I think as a steam fitter…a sheet metal worker. Then he came to Canada and thought that he would like to further his education, and he became a student. Dev Chowdery had a memory that one could only marvel at. Many is the time when I would ask him a question and he would give me an answer and I would say to myself, "My goodness, he's got that dead-on. I couldn't have put it better myself." Only to realize later that that was exactly the way I had put it. His memory was perfect. Then, when it dawned on me that I was dealing with a guy with a memory that good, I asked how was he at analysis and thinking for himself? I would make him, I guess the phrase would be, walk the plank, like the pirates of old. I would ask him a question phrased to sound, superficially, like a question he had heard before—but it was different, and should have somewhat of a different answer. But he would run through his memory, find a question that had been asked before, and then supply the answer to that question.

HAIGH: There's a name for that. In artificial intelligence that's called case-based reasoning.

KAHAN: Oh yes. Well, then I would ask him another question. He had committed himself to an answer that isn't quite right. So I'd now ask him another question that I had, in fact, asked before; he would give me the answer that we'd had before. But now this was even less suitable than the beginning. Finally I'd get him to a point where I would say, "Here is what you're telling me and, yet, you know if we go through a different sequence of questions, what the answer should be. But here is the answer you gave me. Do you see that there is a difference here? Can you explain why there is a difference?" And it finally dawned on him that there was a modality of thought that he did not practice. He had used memory as a substitute, universally, for analysis and reasoning and logic. It wasn't that he was incapable altogether of analysis, reasoning, and logic. He was a clever enough guy but he wasn't used to it. He had substituted memory instead. With a phenomenal memory like that—it was like a photographic memory—why did he need analysis, reasoning, and logic?

What we agreed on was that he was not Ph.D. material. He wrote a perfectly reasonable master's thesis. He understood that research into things, the likes of which he had never encountered before and, therefore, wouldn't be able to rely on memory, was not what he was going to do. He ended up working for IBM as a systems engineer. He remembered everything about the code. If anything happened, he could run through his memory to see what things had fit. So he essentially became an IBM systems engineer in Toronto. He married a Toronto girl, settled down, and had a family. So the last I heard, which was a long time ago, he was happy there and productive—and didn't have a Ph.D. He needed it like a hole in the head.

When you think about it, what do you need a Ph.D. for? You are going to ask me why I got a Ph.D.—I know I saw it on there. Why did you get a Ph.D.?

HAIGH: Because I thought it would be nice to go to graduate school. I didn't look that far ahead.

KAHAN: Well, I got a Ph.D. because I sort of fell into it. I had this master's problem; I thought after that, I would go get a job. But it turned out that I found out all these really interesting things and wrote this paper that almost caught up with Ostrowski. I got some insights into one thing or another and said, "Well, I might as well finish; here I am and it will only take me a little longer to get a Ph.D. thesis." Little did I know that I ended up with several possible Ph.D. theses that I could have written. So I finished one and it was a good one, I must say, much to my surprise. I didn't know when I started that I was going to be able to solve that problem and it seemed to nag at me. I didn't do it because I wanted a Ph.D. I did it because I really wanted to understand why that worked or why it didn't work or whatever. There I was with a marvelous thesis—one of a half-dozen like that, some of them which I wrote up later. Remember I told you about this eigenvalue thing? I wrote that up later, in 1966. I had to go and recover it. I couldn't find my notes, so I had to redo it. It made a lovely paper on clusters of eigenvalues. [W. Kahan, Accurate eigenvalues of a symmetric tridiagonal matrix. Technical Report CS41, Computer Science Department, Stanford University, Stanford, CA, 1966]. Later I wrote up some stuff that ended up in Beresford Parlett's book. It was done because it was a problem that had to be dealt with and I was curious as to what made it so hard. Why are people stumped?

HAIGH: So while we are on this topic, I do have a question. You had mentioned that you had never yourself published your thesis.

KAHAN: No, it was in Varga's book for the most part.

HAIGH: So I was wondering, as you were there as a faculty member and were promoted and given tenure and so on. Did you have to produce a stream of publications?

KAHAN: No. I had to publish something. I didn't publish them in order to get promoted. There were some things that a student wasn't going to do, so I better do it. So some of my publications were little gems. The programs that I produced were little gems at the time. We will come back to some of that. We ask this question--

HAIGH: Were the programs something that a tenure committee would have viewed as something they should take any notice of?

KAHAN: I really don't know. I only know that I was doing what I thought needed to be done, and there were people somewhere who looked at what I was doing and they must have approved of it. I was teaching; I had written a couple of little things here and there; I was contributing to society, the academic society in some fashion that some people must have thought merited promotion to tenure. I don't know because I wasn't on that committee. I don't know how they deliberated. The way you got tenure at Toronto at that time, initially, you would get a letter every year telling you that you had been reappointed, or else it would tell you that you were not going to be reappointed and it would be a good idea to go look for another job. They tried to give you a year's notice. Then one day I got a letter that said, unless you hear from us otherwise, you will be reappointed annually. So I found out one day—Sheila and I opened up the letter and found out, hey, I guess I'm an associate professor now. I don't recall that I did anything to deserve it. It just happened. I must say I like that system better than the ones we have now, where it is an agonizing business. I guess your wife hasn't got tenure yet?

HAIGH: No. We arrived essentially at the same time.

KAHAN: And you have tenure?

HAIGH: No, no, I've only been there a year.

KAHAN: You see, there you have got these uncertainties that weigh upon you, and I don't blame you for being troubled about these questions. I never gave them a thought. It just wasn't important to me. Suppose they fired me…I could have repaired television sets. I just never felt that I was going to be unemployed. I could have worked for General Electric. Canadian General Electric. Or I could have gone off to the United States where there were job opportunities in every newspaper. I didn't worry about it. I was concerned to do what I thought most needed doing, that if not done by me would either not be done or, worse, it would be done badly.

I still feel that way, and I don't recommend that other people follow my philosophy; the world today is not the world I lived in, then. I don't know exactly what the right strategy is for today's world, because it doesn't press upon me the same way it presses upon your generation.

Dev Chowdery needed a Ph.D. like he needed a hole in the head. What's a Ph.D. good for? Well, it's certainly the union card if you want an academic job. But suppose you don't hanker for an academic job. What's a Ph.D. worth? You are going to spend typically four or five years in math, maybe seven years or more in the system-building areas where it takes longer to get a Ph.D. You are going to spend that time and, if you had gone out and gotten that job, you could get some experience and, with that experience you'd work your way up and get some seniority. Now imagine that you spend the same amount of time in industry that you would have spent on a doctorate, and now there are two of you. Two hypothetical individuals—which one is better off, and why? If you've done the sort of work that would have gotten you a Ph.D. academically but you did in industry, you are going to get very comparable respect and station and even salary. Except for one thing: the Ph.D. is worth 20 seconds. If you are Mr. Haigh and you tell your boss something he'd rather not hear, he might just dismiss you right away and say, "Never mind, I told you to do it, so go do it." But if you are Dr. Haigh and you tell your boss something he would rather not hear, he might give you 20 more seconds before he decides to go away and do what you were told. Twenty seconds is what you've got to persuade him that maybe he should reconsider. That is the difference between having Dr. in front of your name or having Mr. in front of your name in industry.

HAIGH: I imagine that is something that will become relevant when we talk about your consulting career?

KAHAN: Could be, yes. But as far as Dev Chowdery was concerned, it wasn't going to make much difference.

HAIGH: So tell me about the other two.

KAHAN: Okay, let's look at Derek Corneil. Derek Corneil came on board…he wanted a degree. His main interest wasn't really numerical; it was just that numerical was what most computing seem to be at the time. He wrote a very good master's thesis on what's called Jacobi Iteration, which I've mentioned, ideas which recur from time to time: people keep on rediscovering them. If they looked up his master's thesis at Toronto, they would find that there it is already. Then he went on to other things. He was really more interested in combinatorial things. He got his Ph.D. subsequently on something altogether else. We still remember each other and our relationship seems to be fairly warm. He was chairman of the computer science department at Toronto for a while. Last time I encountered him was not too terribly long ago, and we were happy to see each other. I don't think that he, nor I, imagined that he was going to be a significant figure in the world of numerical analysis. So the master's thesis, which is a way of getting a master's degree, produced a person, so to speak.

Now Brian Smith was something else because you've encountered Brian Smith in the discussions about EISPACK.

HAIGH: Yes.

KAHAN: Brian was in a rather interesting situation because his elder brother, Casey Smith—Ken Smith—had been a contemporary of mine. We had gone through in the same year, except he went in engineering/physics when I took math. I think he went to Illinois for a while. I think that's where he may have gotten his Ph.D. Certainly Casey Smith's path and mine crossed a couple of times later. Amicably, there is no problem about that. I think Casey Smith is interested in soft computing now. I remember seeing his name, but maybe it wasn't the same Ken Smith—after all, lots of Smiths, and lots of Brian Smiths, too.

Brian T. Smith came to me as a graduate student with a really serious problem: he stuttered. The reason that he stuttered was because his older brother, Ken, was quicker and would finish his sentences for him. So my first task was to enhance Brian's confidence in his competency, which I did by giving him a number of programming tasks, one of which was to program the zero-finding scheme that, it turns out, beats anything else by about a factor of about four. Brian's program actually appears in the IMSL library. It may also appear in the NAG library, I can't remember. A program very much like Brian's can be found in Hewlett-Packard's programmable calculator libraries. I have one—if you're curious, I can go and just fetch it. It's an HP-71B. It hasn't been published widely in Fortran, partly because it really should be re-engineered now, and I use it as something for various students. So some of my students have worked on it, but usually they get jobs that take them away for one reason or another, so they don't quite finish. So I always have this problem ready.

The most recent one was a woman at San Francisco State who was getting a dual degree in mathematics and physics and thought that this would be a suitable problem for her. But she soon found that she really wasn't up to it. It takes a certain breadth of understanding beyond programming even to program something for which all the formulas are laid out for you. The trouble is that there is no way to lay out everything. In a piece of well-laid-out software, there are a lot of imponderables if it's going to do something non-trivial, like in this case solving polynomial equations, given the coefficients. There are a lot of imponderables and, after a while, they mount up to more than some people can keep in their heads. Students generally are intolerant of uncertainty. They really want to know what they have to do to get a grade and how long it's going to take, and so on. They can tolerate a little uncertainty, but it gets to be too much and it's more than they can handle. Programming, even when you think initially you've got everything there, when you really come down to engineering the jobs so people can use it reliably and robustly, so it will be portable and all these other things, it starts to pile up these uncertainties. And once you get enough uncertainty, then some people decide they'd rather do something else.

Brian wasn't like that. Brian coded up this zero-finder and tested it on the 7094. He had a previous version that had been done by a summer student, who had done it on the 7090. I remember his name; he is at the University of Virginia: Charles Dunkl. Dunkl, as a summer student, had programmed the first version of this for the 7090. But there was a lot of support missing at that time on the 7090, and so he couldn't do as good a job as we would have liked to do; he did a good enough job to show that this was feasible using an idea that goes back to Edmond Laguerre, who died in 1896. There is an interesting story about that, but I've written about it elsewhere and I won't cover it here unless you want my notes on Laguerre's iteration. It would take me an hour to print one out.

HAIGH: No, I think I will make do without it.

KAHAN: Okay. Brian wrote this program and it worked really well; it used a lot of the things that I had built into the system. It used gradual underflow; it used over/under flow handling; it did automatic scaling and used extra precision where it paid off and was a really nice program. It was written in Fortran IV, for IBSYS Version 12, and then they switched it IBSYS Version 14, and to the new Fortran, and dammit, there were things that just didn't work. But it worked well enough to get a master's thesis.

Now for his Ph.D. thesis, the question was, how long are the zeroes after you've computed them? Because after you've substituted an alleged zero in a polynomial you get something that is a little bit non-zero. He wrote an excellent thesis on finding regions in a complex plane, which are assuredly the regions that contain your zeros. The work overlapped with something that had been published in Germany just the year before. Although the overlap was substantial, Brian had a good deal of extra stuff so I don't think that he had been scooped. The other author was Borsch-Supan; that was his surname. [Borsch-Supan, W., (1963): A posteriori error bounds for the zeros of polynomials. Numer. Math. 5, 380—398]. I think Brian's scheme was, in some respects, better. Then on the strength of his Ph.D. thesis, he got a job at Argonne National Lab, where he worked on EISPACK, among other things. Subsequently, he went to the University of New Mexico in Albuquerque and, ultimately, became chair of the computer science department there. If he hasn't retired then he is just a faculty member who is in sight of retirement. I think he has lived a good life and doesn't stutter.

[Tape 6, Side B]

HAIGH: So that has wrapped up with your students at Toronto. I wonder if this would now be appropriate to go back and talk some more about SHARE and your activities there.

KAHAN: Yes, I want very much to discuss that because we've come to a good point for discussing that.

HAIGH: I just have a few general questions I'd like to ask about sharing your experiences before we go into specifics, mostly because you are just about the only person I have been able to talk to who has any memory about these things.

KAHAN: I hope it's a good memory! You may be taking a big chance, because if you don't corroborate these things you may find that I misremembered.

HAIGH: Well, some of these things I can corroborate. I'm interested to hear the story in perspective.

KAHAN: How did Hirondo Kuki get into this? That would be a good place to start.

HAIGH: I actually have even more basic questions. You've mentioned that you contributed these routines to SHARE. Presumably you were also looking at the catalog from SHARE and the existing routines there. Did you ever take any routines from the library?

KAHAN: Yes, on occasions I would review a program but not very often, as it took a lot of time. I think it's worth explaining something about this process, which is discussed too glibly, about somehow testing and certifying programs in SHARE. The programs in SHARE ran over a fairly wide range, and you have to appreciate these levels of expertise or ingenuity or whatever. First, I’ve already mentioned that physicists learn to model things, so it’s physicists rather than mathematicians who construct a mathematical representation of a physical situation, hoping that it is, on one hand, accurate enough that it will be worth pursuing to learn something about the physical situation and, on the other hand, hoping that it’s not so accurate that it’s intractable, mathematically. But once they’ve reduced it to a mathematical problem, then it takes a certain amount of cleverness of how to turn this mathematical problem into a constructive procedure so that you can imagine constructing a solution in some reasonable sense. It may be an analytic expression, or graphs that you obtain numerically, or who knows.

So you’ve conceived up, but now you’ve got to write a program. Actually it takes a higher level of expertise to write a program that will give you the results that the mathematics deserves at a reasonable price in terms of the resources available to you. You’ve got a certain amount of a computer resource and you’ve got to figure out how to use that and you’ve got a program to complete sometime in the foreseeable future. So now you’ve written a program and you’ve run it on three test cases and it seems to work. How do you know that it’s right? There are two things to contemplate. One is some kind of formal proof that it’s correct, and that formal proof is something feasible, sometime. A great deal of energy is now going into semi-automating the process of formal proof. By *semi-automating* I mean that a human has to figure out the proof’s strategy, but then the verification of the correctness of a proof once you find it, however you find it, can in some cases be mechanized. Not in all cases, and the reason why is that automated proofs are really very, very bad in equalities for the most part, although they’ve been making some progress in Lyon and a few other places. At Intel there is a good guy there, John Harrison. At Lyon they have Sylvie Boldo—I was on the jury for her thesis. And there have been lots of others and people before them who were pioneers in the area; there were people after them but I just mention these two because I happened to have had some close contact with them.

There isn’t any real prospect that we’re going to automate theorem proving. We can partially automate theorem verification, however. In some sufficiently simple areas, we can automate theorem proving. Floating-point is, apparently, not one of those areas. So you can hope to prove something, but the proof can be a thousand times complicated as the program. So it has a thousand times at least the capture cross-section for error. And it is not uncommon for proofs to be wrong. Phillip Davis has done a survey in mathematical proofs and found that something in the order of a third of them have serious deficiencies—a tenth of them, uncorrectable—but, thank goodness, nobody cares about most of those theorems.

Therefore you also have to have testing. Testing requires a level of deviousness and ingenuity and understanding that transcends what is needed to write the program in the first place. One of the really valuable contributions that Cody and Waite made was the ELEFUNCT tests. Those tests were pretty good for their era and they were testing programs, which were very good considering that they were intended to be portable to damn near any machine. They weren’t as accurate as I would like and didn’t have all the properties I would like but, compared with what preceded them, they represented a giant step forward for all except a handful of places, like Hirondo Kuki’s stuff.

Testing is extremely difficult to do competently. In fact, tests are frequently done incompetently. I noticed a letter in there where I said it appears that someone who thought he was testing my square root was actually testing IBM’s; he just didn’t realize it. He should have gotten suspicious when he realized that my square root and IBM’s had exactly the same output even when they weren’t exactly right, and doubly suspicious when he saw that there was a nonmonotonicity which afflicted IBM's program, but from which I claimed mine didn't suffer. So that got resolved and it turned out he was, indeed, mistaken in invoking IBM's program instead of mine. That's a problem with the linker: if you invoke a program by name, and the linker looks and finds that name first in some list, it will then link that one in instead of one of the same name that occurs later in the list. So it is a dumb mistake.

HAIGH: And that testing was done as part of the Numerical Analysis Project within--?

KAHAN: It was going to be we hoped. But it couldn't be.

HAIGH: Before you even get to that level of testing, which I can appreciate has its own issues, reading the original SHARE procedures from the 1950s, it's clear that they were attempting to deal with this. It does say that the author of the routine has got a responsibility to document it, and also, if an error is pointed out to them, they have a responsibility to fix it.

KAHAN: Well, a moral obligation—which I didn't want to breach, so to speak.

HAIGH: Did you have any practical experience with that, where either someone had pointed out something in your routine in the library, or you had found an error in somebody else's program?

KAHAN: Oh, sure, and sometimes I would give them a fix which would sometimes be offensive. I remember in one case somebody had a problem with a routine where there was a special case that didn't work out quite right; I said, "I'll fix it for you by doing such-and-such." I did something pornographic: I used the fact that I knew what division by zero would do, and took advantage of that to fix the code in the minimum possible way, and that was offensive. Divide by zero, you've got to be kidding! So yes, we did some of that and I can't remember them. There were some; there weren't a lot because it took me a long time. I actually worked on some of these things and sometimes the review would be longer than the program, by far.

HAIGH: In general terms, would you say that many of the routines from the SHARE library have made their way into the Toronto's internal library.

KAHAN: Well, no, the SHARE library was one of the libraries which people had access. They could use the SHARE programs and could use it in the same way they could use our programs in our own library. But there was one very important difference: some of our programs could not be used by other people. For example, somebody at the University of Maryland wanted to use some of our programs. It turned out that our programs depended on so many of the changes we'd made they had to adopt our system, which included our accounting system, which meant that their users were getting bills headed, "The University of Toronto" for a while until they fixed it. It's not a difficult fix, but some programs were more portable than others and were better programs. The ones in which I was interested initially, where I was interested less in universal portability than in delivering results of high quality, meaning good accuracy, robustness, etc. and as humane an interface as I could make it. Some of these things weren't going to work on the IBM system as it had been delivered to us. That is how I got involved in SHARE.

HAIGH: So the SHARE library was available to end-users at Toronto as a supplement, really, to the internally developed library. It wasn't that routines would frequently be taken from the SHARE library and added to the internal library. Mostly the internal routines were homegrown.

KAHAN: There were a number of libraries; I really don't know how many. But I know that the SHARE library was our library, it was one of them—we had that. We had our own library programs, and they were documented in one manual. There may have been others; I didn't keep count. Trixie was involved in these things and our users were not told that they should use our programs in preference to the SHARE programs. They were told go and make your own choice.

HAIGH: What was your personal impression of the typical quality and--?

KAHAN: Extremely variable. There were programs that were abominable; there were programs that were nifty; there were programs that set a standard of quality of which I could only envy and try to emulate. They were extremely few. There were obsolete programs by the bucket load. A high proportion of SHARE programs were sent in as substitutes for programs submitted earlier that didn't work quite so well as the authors had thought. And somebody would say you should use mine but that was standard in numerical analysis. Remember I told you that, for eigenvalue calculations, until John Francis's scheme came up, you'd have lots of programs to choose from; some of them would work and some of them wouldn't, and you never knew until you tried them. In fact, even after you tried them you couldn't be sure. And this was endemic in numerical computation and, to some extent, still is if you want to solve some partial differential equations. You may be on your own or you may use something from somebody's library; maybe it will work on yours and maybe it won't. So this is not a closed subject by any means. It's not as if we know everything even in matrix computation.

A couple of months ago I went to a Householder meeting in Pennsylvania and, by accident, I crossed paths with a guy who had been interested in computing the angles between sub-spaces. Sub-spaces can be represented in a number of ways by matrices. So this is a matrix computation. Chandler Davis and I had written the seminal papers on angles between sub-spaces, and they are here somewhere in one of these heaps. You'll find the papers from which an awful lot of other stuff flows; maybe you have it over there, because I know one of the papers is blue. Here, there we are, these two. This is the big paper that everybody cites [The Rotation of Eigenvectors by a Perturbation. III, Chandler Davis, W. M. Kahan, SIAM Journal on Numerical Analysis, Vol. 7, No. 1 (Mar., 1970), pp. 1-46]; this is the little paper that they generally overlook [*Bulletin of The American Mathematical Society*, 1969, Volume 75, Number 4, pages 863–868]. And if you want you can find a paper by Chris Page and a co-author with a Chinese name, which was published, in middle- or late-'90s, which surveys the field. It's very difficult to read, but you get the idea that he was one of our boosters, because he really thinks highly of that paper and subsequent work that emanated from it.

So computing angles between sub-spaces is a fairly conventional thing to do, nowadays. What's marvelous is that the algorithms most widely in use, there are two, lose half their figures in critical cases. The obvious algorithm loses half its figures when the sub-spaces are very close. The remedy was another algorithm, which loses half its figures when the sub-spaces are orthogonal. The composite of the two that tries not to lose figures is so hideous that papers—long papers—have been written about it. This guy's name was Klyuyev—Russian. He had written just such a paper, and I was commenting on my process of computing angles between sub-spaces and had found that it was a somewhat more interesting task than MATLAB thought. The MATLAB program wasn't working well. Klyuyev said, "You should use our program," so he showed me their program, and I said, "That's horrible. Here is the way to do it," and gave him a very simple algorithm by comparison. Of course, because this was supposed to be simpler than his paper, he found he had to resist it at first but, the next day, he acknowledged it and said he didn't realize it was that simple. See the date on those papers—1970? And it's now 35 years later, and they still haven't gotten the hang of computing these things in the obvious, simple, accurate way. Numerical analysis is full of this stuff.

There are people who berate me when I say there is a really simple way to do that and they say why haven't you published that? I say that it is posted on my Web page; of course I don't have it in neon lights or something, like "come here to compute such-and-such in the obvious way." You have to read some things to find them, and I'm not the only one who knows who to do these things. Numerical computation is subtle because the two malevolencies—round-off and over/under flow—have no names that you can put in your program. If you name the rounding errors in your program, give a name to every one, you are going to drown in names. If you try to put names to overflow and underflow, you are going to find that any effective result from putting those names in is going to get you trapped into the operating system and it will take you forever to get out. That is part of what we are working on our committee right now—how to deal with over/under flow in a reasonable way so that you won't spend forever on it.

The real point is that this stuff is subtle. Testing and consequence is a subtle matter; it is easy to test things badly, very difficult to test them well. To give you another example of that there is what is called the UCB test. It is the name of a family of tests in the library that David Huffman names on his Web page. It is part of FDLIBM, or it's in the same library as FDLIBM. It is a set of programs that tests the accuracy of the elementary transcendental functions in a way that Cody and Waite could not have used because they wanted their codes to be portable to perverse machines as well as reasonably clean ones. So their ELEFUNCT tests are relatively crude tests by comparison with the UCB tests.

UCB tests were coded by Alex Zhishun Liu. He was a master student under me and got a Ph.D. from Beresford Parlett and works now for Sun Microsystems down in the South Bay somewhere. He wrote for me a set of test programs whose job was to estimate the error in the elementary transcendental functions on certain important subsets of their ranges to within a fraction of a unit in the last place. The ELEFUNCT tests can't possibly do that. They test to within a few units in the last place, and this is not to disparage them. For their time, they were marvelous. For their target they were just right. Nowadays, we would like to have better programs and Alex wrote these tests so that he could run these tests on the library that we were developing for 4.3 Berkeley UNIX. We wanted to be able to show that they were very accurate. So these tests used continued fractions and all sorts of very, very devious programming tricks. Alex Liu can paper the walls of his bathroom with unsolicited testimonials from people who say thank you for your test—it discovered such-and-such a bug in our hardware, or such-and-such a bug in our compiler, or whatever. They really do exercise the hardware, the compiler, and everything in order to run these tests, and they are pretty fast.

We used tests like these, and then some, to test the elementary transcendental functional library that Peter Tang produced for the first Pentium generation. I persuaded Intel to do the testing thoroughly, and they found a couple of cases where Peter's error bounds weren't quite right and Peter fixed them. After that nobody has found any bugs in the transcendental functions on Pentiums—at least if they have, nobody told me about them (and I would think that they would come to me with their hands waving, but it hasn't happened). These are really good tests; they are very, very difficult; they have never been fully written up because the tape is too long.

So I want you to appreciate that conscientious and competent testing is extremely difficult even for the elementary transcendental functions, and for them tests are easy by comparison with testing other codes. Testing matrix codes, for example: we have been spending at least three years on testing some codes on some things that are going to go out this year, I hope. Testing partial differential equation solving codes—I'm not sure that anybody has a decent idea about how to do that in a reasonably reliable way. Somebody competent to do a reasonably credible test might not be so easily found and if you do find him, he might have other things to do with his time.

HAIGH: So let’s get back and talk about that process, then. As I understand it, by the early 1960s, there was a perception that the quality of routines in the SHARE library was sufficiently low that it compromised the original purpose for having the library. Many installations would rather write their own code than taking a chance on taking something from the library and testing--

KAHAN: Well, actually many people would actually like to write their own code than using anybody else’s regardless and that was a dominant factor.

HAIGH: Well, that’s true. But anyway to make the library more useful and more credible, it would be necessary to weed out the bad routines and keep the good ones and improve--

KAHAN: And they found a way to do it, which was to reissue. So codes that nobody really wanted to stand up for simply didn’t get reissued.

HAIGH: And did you mention that was with the--

KAHAN: That was the 3000 series of codes.

HAIGH: That was associated with the transition between machines, wasn’t it?

KAHAN: It was associated with the transition from the 7090 and the 7094, but there were still some 7090s around and there were a lot of 7090 codes that still worked very well on the 7094, of course. It had to be. The main thing was that the transition from the 7090 to the ’94 provided an excuse to ask for reissues, and if there was nobody who wanted to see a code reissued, then the code would die. Many of the old codes for the 704 and the 701 (I don’t know if there were many for the 701) died a natural death simply because nobody wanted to reissue them. Many of them had been superseded by a presumably better code, and so they disappeared from the SHARE library.

HAIGH: As I understand it, the Numerical Analysis Project was designed to essentially to provide the same kind of peer review mechanism that these numerical codes--

KAHAN: Ostensibly, yes.

HAIGH: That people had been using for a long time with academic papers.

KAHAN: I know that Hirondo tried heroically and Bruno tried heroically to find people to review programs, but it was just too difficult and too expensive. The managers wouldn’t pay these guys to review the codes if they were being paid, instead, to design guidance programs for missiles or the spread of explosions from personnel anti-personnel mines or armor piercing shells or whatever it was. So you couldn’t get reviewers who had the time as well as the competency, which was true even in academia. In academia—how should I put it—a review doesn’t earn you as much in the way of brownie points in the way of a published paper. People who are sensitive to that would have found the reviews were not all that productive of prestige or whatever it was that they aspired to. Reviews could embroil you in disputes and could be very time-consuming, so it wasn’t always that easy to get competent people in academia to do reviews. I think there were some heroes who tried, but the project was doomed because there wasn’t a way to ensure… ideally, the person who submitted the program and who wanted to get credit for it should have been willing, in some sense, to pay to have it reviewed by an anonymous review. Of course, this isn't a feasible mechanism. I think even today we just don't have a good way to deal with these things despite good intentions.

That was overshadowed by something else that happened which made all this issue about reviewing the quality of submissions to SHARE irrelevant.

HAIGH: Before we jump off reviewing, I have two follow-up questions. In practice, was it just the case that the volume of available reviewers was just completely dwarfed by the amount of reviewing that needed to be done?

KAHAN: Yes, willing and able, that's right. The volume of willing and able reviewers was inadequate for the task.

HAIGH: Right. The second one is, as I understand it, a very similar review process was put into place a decade later with the start of the *ACM Transactions on Mathematical Software*.

KAHAN: It was actually earlier in the *Communications of the ACM*. It had an algorithm section and they submitted things to reviews. Many of my programs got reviewed that way.

HAIGH: Is it your impression that the reviewing process in those arenas was working better than it was in the program library?

KAHAN: Not noticeably. Many of the reviewers would simply run some tests and said they had run it on these tests and it had worked and that was it. Other reviewers would really try to be critical and test things carefully and they would end up, in many cases, writing a review, which was substantially longer than the submission.

HAIGH: I also wanted you to talk about IBM's SSP; I know that was one of the big things that they were attempting to review the quality of.

KAHAN: The original SSP was a marketing device. It was designed to show people that IBM could supply you with various codes to solve some of your more difficult programs if you can't program them yourself. But the SSP programs were not necessarily written to a standard that we would consider respectable, nowadays. They were stopgaps, and it's easy to criticize them, because they had so many flaws. But to be honest, they didn't have a hell of a lot more flaws than were commonplace in programs that people wrote for themselves. If you read books like *Numerical Recipes*, first of all, *Numerical Recipes* was as ridden with silliness as SSP. Their first edition was ridden with crappy stuff, and their second edition is somewhat better, but still flawed. I told you that you mustn't think of software an inventory. It's the minds of the people who understand it and, of course, understand its limitations. Or at least they're interested enough that they are going to pursue limitations, should such become available. That's what's valuable. The deck, the floppy diskette, the image on a disk—these things have a certain value, but it's not as much a value as you'd think.

HAIGH: Do you know who in IBM had been responsible for producing SSP?

KAHAN: No, I can remember some names that were associated with it but I don't know if I could call them responsible for it. Ed Battiste at one time had some connection but I don't know what it was. I think he was really in the statistical end of things. I don't know who was responsible for the other programs and, in any event, it was customary then and alas still now, for programs produced under the imprimatur of a corporation to be presented anonymously. You never did find out who wrote the programs. I'm not sure that the programs were actually written that way. It's possible that these were programs collected and somebody's job was to produce a more nearly uniform interface to the programs rather than rewrite them. I know that that happened often, not just in IBM, but in other places.

HAIGH: It seems that at least some members of IBM and the numerical analysis community were quite vocal in expressing their dissatisfaction with some aspects of the package?

KAHAN: Well, yes, you've got a package with a program in it that purports to do a problem that you think was your baby and you look at it and it's easy to see, my goodness, this is an ugly baby—nowhere near like mine! So it was easy to criticize. Now the question is, what are you going to do about it? Can you produce something better? My policy was to produce something better whenever something better could be produced, and that's what I did. That's what brought Clemens Roothaan into the picture.

HAIGH: Before we shift away there, do you have a sense of whether IBM did a good job in later versions with fixes of rectifying the biggest problems with SSP?

KAHAN: It varied. I know that some of my friends at IBM Yorktown Heights were active in supplanting SSP programs with better ones. There was Fred Gustafson, whose name I've mentioned to you, and he had some friends who helped. IBM must have been of two minds about this. The bulk of the benefit that IBM could receive from improved SSP might be to enhance IBM's reputation in quarters where there weren't very many people to sign the checks for buying IBM equipment. Each company was supposed to have its group of numerical experts. In many cases these experts are supposed to provide assistance to the marketing people who may have a potential customer and would like to know that customer's job load will run satisfactory on the computer and they'd like to take some numerical analysts over to look over and give them an opinion. The numerical analyst will say that this is a job that is going to take some months, and the marketing people will say I don't have that much time. I'd like to know sometime tomorrow or next week at the latest.

So DEC, for example, had some very good people. Mary Payne comes to mind. She had some others working for her or with her; I don't know exactly what the arrangement was. They were quite competent and also very opinionated. CDC had people in whom I had rather less confidence. Cray finally got somebody who was competent in the arena. IBM had lots of people with diverse competencies, but they were spread around the corporation.

They were not necessarily going to produce the best value for IBM by improving SSP, because the SSP package was not, so far as I can tell, intended to be used as a production vehicle. It was intended, so far as I can tell, to justify statements by their marketing people that IBM's programs could do various jobs, but maybe not very well. It was like shooting fish in a barrel to find flaws with that package, and it wasn't the first one.

One of the papers you have in here is about anomalies in the ACRITH package. Well, you will see how many goofy things we found. I tell you when you look at somebody else's software, it's sometimes all to easy to find lapses. It sometimes costs a life, not in the sense of being blown up in Iraq. But rather a life devoted to understanding something well enough to write a really good program that, then, advances civilization in the best way—which is, of course, that people can benefit from your experience without having to relive it.

HAIGH: You had another issue that you wanted to talk about which I suspect is going to turn out to be the problems with the 360 arithmetic.

KAHAN: Yes, but first, Clemens Roothaan. I became active in SHARE and I wanted to achieve various reforms in the way some things were handled. I discovered that there was a kindred spirit on the other side of the room in the SHARE meeting—that was Gio Wiederhold. Well, Gio has an interesting history that deserves to be told. He was born in Indonesia. His mother had lived in Indonesia; her family had been there for some generations, of Dutch descent. For some reason she was in Italy in the Second World War. She married a German officer, which is why Gio has a combination of an Italian name and a German surname. His father seemed to have disappeared from the scene. I don't know why. He may have been killed; it was wartime. His mother ended up back in Indonesia, or so she thought. But Indonesia was one of the few places where Roosevelt's policy of denying European powers their right to reestablish their Asian colonies was enforced was, because the Dutch were too small and they could be bullied. The policy was not put into effect in Vietnam, alas. It was not put into effect in India, but India was a special case because the Brits agreed to grant Indian independence. It was not put into effect in the Philippines, but that was a special case because the Philippines were soon granted independence. Elsewhere, Vietnam was the festering sore.

Unfortunately, Gio's mother was expelled by the Indonesians, who wanted not only to get rid of Dutch colonialism but to get rid of the Dutch. Alas, she was not accepted in Holland because she was not a Dutch citizen. She was of Dutch descent, but not a Dutch citizen. So Gio and his mother found themselves stateless. How they managed to get into the U.S. I don't know, but definitely Gio did not have a green card. He was working as a programmer for the Berkeley computing center in the early 1960s while I was at Toronto. Somehow he would get the same ideas as I would get. So from opposite sides of the room, and not having met, we would both be on the same side of a substantial number of issues involving technical reform. And that's how I first met him.

HAIGH: Did you ever serve as the Toronto representative to SHARE?

KAHAN: I was *a* Toronto representative to SHARE; Trixie was *the* Toronto representative to SHARE. I would go to some of the meetings—the ones that were reasonably accessible, or I had the time, or it wasn't during teaching season. My expenses on those trips were paid by the University of Toronto. They were repaid, in many respects, by the fact that I was a teetotaler. So during happy hour, I drank ginger ale while the others drank something stronger and then boasted of things that I know they shouldn't have been talking about—but I got some idea of what was coming down the pike.

HAIGH: The so-called SCIDS--

KAHAN: That's right, yes, they were called *SCIDS*—and people really did skid. I was listening—no, I wasn't listening, I was eavesdropping. That was important to me because we have a long pipeline and I have to prepare students not for what's the problem now but what the problem is going to be when they get there. There is a certain amount of gambling involved, and any extra information helps.

I had contributed a number of revised IBM elementary functions to SHARE and, on one of the occasions there, Clemens Roothaan asked, "Why don't you just rewrite them all?" I had to explain to him that I was a professor and I had other duties. I could only rewrite these things as the occasion arose, and even I wasn't really out looking for trouble—I was just trying to fix the things that bugged me. He said then I will take care of this and he got Hirondo Kuki into the case and that is how Hirondo got started. Hirondo did an excellent job, and once I saw the quality of his work I realized that I don't have to do this anymore. Hirondo wrote codes sufficiently good (not just for the 7090/7094 but, also, if memory serves—for the 7044 and for some other machines) that he became engaged in a contract to write the transcendental function library for the IBM 360 series that was coming. That would have been in 1963, I think…but we didn't know that. What we were doing in this group was looking at a number of proposals to make Fortran computation better. Some of these proposals would come from various sources to enhance the compiler. The proposals that most concerned me in early 1964 were the floating-point proposals from that document which, though it's dated in 1966--

HAIGH: This one?

KAHAN: Yes, systems support. Most of the things that are in there were actually up and running in 1963/early 1964. It may have been that they were all up and running; I just can't remember. I had started to accumulate very considerable experience, some of which is documented in there, that these things were the way to go. So I was going to persuade the SHARE committee, the numerical analysis committee or project, that this was the right thing to do. I think they were persuaded.

We had a spot on the program for the SHARE meeting that took place in San Francisco in March of 1964. In the Jack Tar Hotel, which I think suffered fire and is now called the Miyako, or something…it's got a Japanese name but it's still the same place. It stood up on a hill and everything. After we got our program ready and our presentations and everything, we were asked by IBM if we would be willing to take a different venue; they wanted to use this particular room and time slot for a special announcement. We didn't want to be intransigent so we accepted a shift or a change of the schedule.

We had been ambushed. That was the meeting at which they was announced the IBM 360. Once they announced the IBM 360, no sessions about anything else were populated. Everybody wanted to go to the sessions involving the IBM 360. So we had nobody turn up for our discussion about exception handling and Fortran, floating-point exception handling and Fortran, or anything of the sort.

There was a planners' session where Hirondo Kuki explained something about the transcendental functional library for the IBM 360, which was what he had been commissioned to build. We thought we knew Hirondo Kuki, but we didn't. His English wasn't very good and so, when he gave his talk, we thought he was boasting about how cleverly he had written some of the programs. That's not what he was doing. For a cultivated, refined Japanese, boasting was really not considered cultivated and refined. He wasn't boasting. He was trying to alert us to the fact that there was something very wrong with the arithmetic. And we thought he was boasting, so we paid no attention—until November; then we got the principles of operations manuals and we saw what he was talking about. The arithmetic was perverse. Cody has mentioned something: if you multiplied a single-precision number by one, it just gave you that number. But if you multiplied double-precision number by one, it lopped off the last hexadecimal digit, because it didn't keep a guard digit in multiplication. If you did a subtraction in double-precision, it didn't have a guard digit; it did what I complained about in the Cray document. It was hexadecimal and it was chopped in a slightly perverse way. Not terribly bad in single-precision, but it meant that the arithmetic that you were used to wasn't going to work anymore. It wasn't binary, it wasn't rounded, and it wasn't chopped the way you are used to so something else was going to happen. All it did was shift something and it didn't give you the same result as if you multiplied by a half. It did a shift without a guard digit. We went nuts. Suddenly we realized at last what Hirondo had been trying to tell us: there was something really perverse about the arithmetic.

This is when a committee was formed and-- one of the few times in my life when I acquiesced to joining a committee. I have already told you what my views about committees are. This committee had Hirondo, me, and a statistician from Maryland or Virginia whose second name was Williams. It had Len Harding, who was a graduate student at the University of Michigan, whose interest was in arithmetic hardware. Whether Len Harding ever got his Ph.D., I don't know. He wasn't the sort of guy who would finish a thesis, but he was knowledgeable about hardware at Michigan. If I remember rightly, Pat Sterbenz was an observer. I think I've got everybody.

HAIGH: At the time this committee was formed, had anybody had in actual experience with 360, or was this just something that you were realizing as you looked at--

KAHAN: We hadn't had a 360 to play with ourselves except for Hirondo. He had something which to test his programs but the rest of us hadn't gotten delivery of 360s yet. We could see they were coming, and we realized that something really bad was going to happen. We realized that the programs that people had written in Fortran, if recompiled, would malfunction on this machine. The stuff I've given you about the Cray users group, somewhere, it tells you how programs written for other machines would malfunction just on Crays; we could see this happening to…it wasn't just the SHARE library. We could see this happening to almost everybody on the 360s, so of course we were desperate.

And nothing we said to IBM seemed to make the slightest difference. There was not the slightest inclination in IBM to do anything. As far as they were concerned, the arithmetic was fine. They had a couple of papers to justify their decisions and we got nowhere. So by 1966, our committee decided that, since IBM was not going to change the hardware and deliveries were being made, the time had come to figure out to make the best out of a bad situation. We prepared a tutorial: Problems Converting Your 7090 Codes to 360. I was going to do the presentation. The committee had contributed from various places had collected some examples. We had analyzed as many codes as we had time to analyze to see where troubles would occur. We prepared transparencies. I wish we had those transparencies now.

So there it was, August of 1966, and we made our presentation. Something strange, something really funny was going on. For one thing, we were moved to a bigger room because SHARE would get an indication from its membership who was coming and what they wanted to attend. For some reason we ended up in a big room. In our past SHARE meetings, we would be lucky to get six people…to try and explain this terrible difficulty. "I've got to do something, get IBM to change it, quick before it's too late!" Six people. Now, we had this huge room—and it was full. The lights went out and I start droning on about my presentation and there are these expletives coming out of the audience: "How could they do this?" "Do you mean to tell me…?" "So that's what happened!" All this sort of thing was coming up while I'm trying to present, in a calm and reasoned way, what you have to do. Here is how the code used to be written for the 7094; this is what it used to do. If you run this on the 360, this is what will happen instead. They had put in something for D-PRODS so that you can accumulate products to extra precisely, but you have to remember to write this into your code, and so on. If you want to take this type of code on the 7094 and do it on the 360, there is no way. You are going to have to write some assembly language-something-or-other in order to accomplish this, and it's not going to happen in Fortran and so on…and on, for a reasonable period of time—like about 40 minutes.

Then it was time for questions. The lights go up and there are all these red faces with angry people. And these questions kept coming up, "Do you mean to tell me…" "How the hell is somebody supposed to do such-and-such?" I'm trying to explain to them, "Look…we tried to warn you about this. We've sent stuff out through SHARE membership and so on, and we were trying to tell you this." But what we hadn't figured on was that this meeting was in a foreign country; it was in Toronto. The people who attended weren't the usual bunny rabbits who hop in and out of operating systems and stuff. These were their managers; they had come with their wives; they were going to see Niagara Falls. These were not guys who were programming and looking at the various SHARE messages and so on that we were sending out. These were the guys who had to pay the programmers—and these were the ones with the red faces.

What about the ones with the white faces? They were the guys in IBM's standard summer uniform: sort of a gray linen suit. They were the salesman and had come because they knew the managers of the guys who, in effect, signed the checks—or else they were the ones who signed off recommending that the check be signed. They were coming there to be with the managers. But why had they turned up at our session instead of the other sessions? So we went out and looked at the various boards loaded with acronyms. Ours was in English; that's why they had come. We heard about this afterward. That was August and, by late September, we were told that the IBM folks in the Time-Life Building would like to see us. The meeting was scheduled for October or November; I can't remember which.

We turned up there, four of us. Sterbenz was just an observer, so there were just four of us on the committee and we were sitting on one side. And on the other side were the various IBM types, and there was the chairman of the meeting somewhere in there. We thought at last we are going to present our case and get them to change the machine before it was too late. No, that's not what they wanted to know. They wanted to understand what tests we had performed. How had we known to analyze this-or-that in order to find out so-and-so? They had not the slightest intention of changing the machine; they just felt that, for the next design phase of whatever, there must be some technology that we had used that they didn't have—which, of course, we did have. We had experience which they didn't have.

In the middle of the morning, the vice president of SHARE in charge of applications turned up; he just wanted to see how things were going. As he left the meeting he stood behind us and put his hands on our shoulders collectively and said, "I want you to understand that, as far as SHARE is concerned, what these guys say goes." And suddenly we found ourselves holding all the cards. So after lunch we were meeting with a different team. Must have been some very hurried discussions, and now we were meeting with people who were going to discuss, seriously, the various changes.

I presume I'm not telling you just what you want to know. The greatest impediment to change, according to the senior guy sitting in front of us (I think his name was White) was that IBM was the greatest publisher in the Western world—they had published more stuff than anyone else. If we made these changes, they'd have to change all the manuals. We were thunderstruck. This was their big objection? Well, we got down to cases. Changing hexadecimal back to binary appeared to be out of the question. I regret that, because it was a truly backwards step to choose hexadecimal, but they appeared to be adamant for whatever reasons. We didn't find out until later that hexadecimal worked faster on the small machines, which were going to be their bread and butter. On the large machines, going back to binary would have been a good thing—better performance and so on. On the small machines, which were mainly micro-coded, going back to binary would have been a tough act. It would have required that they build data paths in there that they hadn't provided. Getting rid of some of the other difficulties was a feasible thing, although it was going to be expensive.

Hirondo Kuki agreed to write two more versions of the elementary functions library. He had already written a version to run as well as he could get it to run on the crappy hardware. He agreed to write a new version that would work on hardware whether it had been changed or not. It would at least give decent results. And then to write a third version, which would take advantage of the removal of some of the things to which we most objected; it would get the best results the fastest; it would be the ultimate library (of course, nothing is the "ultimate library"). It would be the library that we would have to go with a machine with the revised hardware. If you want to find out about how the hardware was revised you'd have to get principles of operations manual printed in 1964–65 and compare it with the operations manual printed in 1967–68.

HAIGH: I also come across some discussions of the specific changes in the files from the numerical analysis project at the National Museum of American History. So they have some correspondence that goes with this, and I think some of the documents that they have there would be the summary of the changes that would be distributed to SHARE members. Also there is some discussion of the issue in the committee summaries included in the SHARE proceedings volumes.

KAHAN: Yes, of course; there was a lot of documentation there.

HAIGH: So by the point that you had your meeting, had an appreciable number of 360 machines actually been delivered to customers?

KAHAN: Oh yes; it was going to be expensive. They estimated it would cost them something of the order of one or two million dollars. We thought that wasn't far off. We felt we were helping because Hirondo was going to rewrite the library. We were going to prepare material for SHARE members so they would understand why there was this transition. At that meeting, IBM said to me we would like to have you come and visit us for a few months. That would have been the summer of 1967. Hirondo they had under contract. Williams was at sort of a captive IBM shop. Len Harding, at Michigan, was at a captive shop because they were working on the time-sharing version, 360 model 67. So I was the only loose cannon, and they wanted me to come to IBM. They thought Yorktown Heights would be okay, and then I would come and visit some other places. I said I've already got things to do this summer; why don't I visit for one week each of my summer months, and I'll do this on the condition that I can visit the people who are doing the exception handling. Because remember what our original goal was in1964—and I still wanted to get the exception handling turned into something decent. That was the agreement.

But of course people renege on their agreements, don't they? IBM suffered terrible costs. It was not $2 million; it was some modest multiple of $10 million.

[Tape 7, Side A]

KAHAN: Why were the costs so much higher? Well, we should discuss that, and then we'll discuss the things I did that summer. The way IBM was hurt most by this episode was that IBM's customers, who had previously regarded IBM as infallible, now realized that IBM wasn't infallible. Previously, if a program misbehaved, the fact that the hardware was at fault would not generally cross their minds. But now it did cross their minds, and in spades! So when the customer engineer out at some installation obtained the new boards and scheduled the replacement for some evening, the next day, people would be complaining that programs that had worked before were now malfunctioning. The new boards must be wrong and, for all the engineer knew, they might be right. So what would he have to do? He'd have to replace the new boards by the old ones and rerun these programs and discover that they didn't work with the old boards, either. Then he's got to put the new boards back, and he'd have to do this a few times at each installation. Of course it cost tens of millions of dollars when you reckon up their time for this. The cost of the hardware was the least of their worries. It was the cost of this back-and-forth. Throughout IBM's experience, people were now questioning the hardware in a way they had never questioned it before. This brought up all kinds of costs. I mean, you could call them help-desk costs. That's why when some IBMers would see me, for years afterwards, they would just shake their heads ruefully. It was a mistake that they hoped they would not repeat but, of course, I can't tell you that they didn't repeat the mistake. This was a wound that festered in IBM for quite a while, and the next occasion for that wound was the release of the IBM PC. But that's getting far ahead of us.

HAIGH: So in terms of the effect that this had on your own career, it seems that looking at the studies you later produced of problems with Cray and with Java and those other kinds of things. Would you say that this experience emboldened you to think that, if you made the case sufficiently persuasive and sufficiently well argued that the world might take notice and fix the things?

KAHAN: Yes, that idea crossed my mind, but actually, you see, I didn't really want to get involved with this sort of thing again. When I went to Berkeley and found that I was involved in this sort of thing again, it wasn't as if I'd been looking for it.

HAIGH: So, we'll talk about that when you get to Berkeley.

KAHAN: Yes. But, no, all I wanted to do was solve differential equations; if my program didn't work, I wanted to understand why. And when I would discover that the reason it didn't work was because the hardware did something that no reasonable man would have expected it to do, then I thought, "Well, what shall I do?" I could either program around it, which is the first thing you tried to do. Or if programming around it gets just simply too horrible, you've got to wonder whether it's going to make some other change, somehow. You know, to go looking for trouble: "Let's look at various machines and see how many bugs we can find". Well, I don't think I'd even want to be paid for that job. I worked that summer, the summer of 1967, on things that I had wanted to do, which mainly involved differential equation solving.

And I went one week each month to IBM, stayed in a motel near Yorktown Heights, and visited and made friends there. I'm really pleased with the friends that I made there. One day each week I'd go to Poughkeepsie, and we would discuss what I thought was a hypothetical study to see how you would build what we now call quadruple precision, what they then called an extended precision arithmetic, with instead of 14 we had like 28 significant hexadecimal digits. So I discussed that with the engineers to know how you do it, the different methods. I don't know why they thought I should be so knowledgeable about hardware, but apparently I was. Then I remember the last meeting was a meeting in August, and I said, "Our next meeting should discuss division." But they said, "No, we think we've got division okay. We understand division now." So, we never did have a discussion of division.

I think it was October or November when I discovered that we were talking about the 360 mod 85. It came out with this kind of arithmetic. It did what I figured it would do except for division. They didn't get division quite right. Division should have been chopped in order to run the same way as division at the lower precisions worked, so that the programming technique could be homogeneous. That is, whatever reasoning you'd used for double, if you just translated all your variables from double to extended and changed your constants and all, the program should be work in about the same way but with more digits left over afterwards. Now, division didn't quite work properly, so we should have discussed it then. I shouldn't have acquiesced, but I thought it was just an academic exercise—sort of give the troops practice. I had no idea they were going to build a product. Later I discovered what was going on by eavesdropping at a SHARE meeting. It appears that NASA had had trouble calculating ballistic trajectories and orbits, and they had figured that the double precision available was not enough. The double precision available would give them 14 hexadecimal digits, which sounded like 56 significant bits but it wasn't. It was actually 53, because of this wobbling precision. And they had decided that wasn't quite enough. They put out an RPQ for something with more precision. Univac had a double-precision package programmed essentially in fixed-point arithmetic to simulate double precision with 27 plus 35 significant bits. What does that come to, 62?

HAIGH: Twenty-seven plus 35? That sounds plausible.

KAHAN: Okay. And the folks at IBM did not want to lose a sale to Univac, so they reckoned that they would really kill the problem but good by building this extended precision. Of course, I didn't know that at the time. What's more, I don't know it for a fact now because it was scuttlebutt. It was rumor, you know? The people who came to SHARE included both the IBM types, who knew what future products were around, but they were under some constraint not to talk about future products unless they were drunk, which fortunately did happen.

HAIGH: Yes, they had legal problems with pre-announcement, I think.

KAHAN: Oh, did they. Yes, indeed. I was thoroughly familiar with those in 1967. They were weighing on people's minds. We can just digress for a moment. IBM was forced to divest itself of its applications group, people who gave advice and did applications programming for other folks. They divested themselves of that. It went to CDC, which improved both the Service Bureau--

HAIGH: The Service Bureau Corporation.

KAHAN: Right! How do you know all those names? I guess you must have been immersed in this stuff. Yes, the Service Bureau Corporation left IBM and went to CDC, and raised the tone of both companies. So people were…how shall I say? They were a little bit gun-shy about announcements, but once they'd gotten enough liquor in them, they could say anything. You never knew what you'd get, and the same thing is true for the guys who work for the Jet Propulsion Lab and various others—NASA, or I guess the precursor of NASA. I can't remember what it was called. I guess it was NASA because *Sputnik* went up in '57, didn't it? Yes, so that was NASA. So I'd eavesdrop on their conversations during SCIDS, and this is how I heard, putting bits and pieces together. This is what I put together, but of course I might be wrong. After all, if you assemble a jigsaw puzzle without some of the pieces, you just might get it wrong. But as I understood it, it was NASA's request for a machine that would have more floating-point precision in order to compute these orbits properly that stimulated the demand for this higher precision. But you know what happened? They hardly sold any 85s. They sold some, but very few. And speed in extended precision didn't seem to matter. So in the rest of the line, instead of building this extra precision into the hardware, until they came to the 3090 series, if I remember rightly, they did it in either micro-code or did it in software. It doesn't matter. It didn't have to be terribly fast because speed didn't affect many of the sales. Hardly any of the people who bought the stuff cared about this higher precision.

HAIGH: Right, and if you would only use it occasionally, then someone doing it in software wouldn't be--

KAHAN: Well, that's the way you might reason. It's faulty reasoning, but that's the way you might reason. And the same thing happened to DEC, you see. DEC later released the VAX somewhere around 1978 or something like that, and they found that they weren't going to get customers to part from CDC unless they had a wider exponent range than the ten plus or minus 38. So they came out with a G format that had an eleven bit exponent that got them to ten to the plus or minus 308. That was more or less comparable with CDC, and then they found there were customers who felt that unless they had this extended precision, they couldn't really compete with IBM for sales. Not that it really mattered, but there were some people who thought it mattered. So the next VAX came out with an H format, which is 128 bits wide, and it had an exponent range around ten to the plus or minus 4,900. And you know what they discovered? The speed of the H format didn't seem to affect the sales. Maybe a few people cared, but hardly any. After you've sold four machines, what're you going to do with the rest of them? So, that was an interesting exercise, and it's very relevant today because our standard committee is trying to standardize on quadruple. We would like people to build quadruple into the hardware, but that's a story we'll come to later. That was part of what I did that summer.

Another part of what I did that summer was to make the acquaintance of some clever guys like Fred Ris and Fred Gustavson. Fred Gustavson was a relatively new hire who was inarticulate. He's inarticulate today, too, pretty much; but very, very clever. He's a counterexample to a principle that I hold dear, which is that good programmers are generally good programmers because they are well-equipped to handle language—that they're good at mother tongue as well as programming languages. But Fred Gustavson is a counterexample. He had written a marvelous program to solve sparse matrix problems. Sparse matrix problems are problems where the number of unknowns is gargantuan, but each unknown is connected in equations to only a modest number of the other unknowns. And solving sparse systems was a high priority item and still is.

HAIGH: Al Erisman discussed that in his interview, and I'm talking to Iain Duff later this month.

KAHAN: Well, Iain Duff is the right guy for that, yes. What Gustavson did was to turn the programming paradigm on its head. Usually you read data from one tape, do something to it, and write it up on the other, so you're program stays accurate. He reversed the paradigm. What he did was translate the program into a straight-line program with no branches, which was valid for some sparse problems and, certainly, for many of the ones he had to be concerned about. And he'd store that on tape, and the data he kept in the memory. Thirty-two thousand words? That meant he could handle, perhaps, something on the order of 30,000 variables, which is quite a few, maybe 20,000 variables. Then he would squirt the program from the tape through the buffers in the machine, and while he was executing out of one buffer, he was loading into another. And executing in that buffer would go just as fast as it can go. So he was solving these sparse problems at a ferocious rate compared with everybody else. That caught my eye. I remember saying to Schmuel Winograd, who was the head of the math department at that time, "You've got a very clever guy, rather cleverer than you may realize, because he's inarticulate." Well, Gustavson's flourished since then.

Fred Ris had just finished a thesis at Oxford, and it was about interval arithmetic, and it was clever, too, in its way. And Alan Hoffman, who was an old hand at matrix theory and matrix computation-- well, I met some really interesting people, so I think on the whole I enjoyed it. But what about the people at Time-Life? Every week I would ask, "When can I see the guys at Time-Life?" And I'd be told, "They say they're not ready for you yet."

Well, it was my last visit, I think it was in September of 1967, and two things happened. One was that I was going through U.S. immigration in Toronto, because that's how it works. You may remember, if you've ever gone up there, that if you fly from Toronto to the U.S., you go through U.S. immigration at Toronto so that your landing in the U.S. can be at a domestic instead of foreign terminal and so on.

HAIGH: I think it's the same in Montreal and Vancouver.

KAHAN: Yes, that's right. Okay, he asked me the question that hadn't been asked at any of the previous meetings, which is, "Are you going to work there?" And I said, "Yes." "Work for whom?" I said, "IBM." He said, "Where's your work permit?" I said, "Well, they told me I didn't need it." And they had told me I didn't need it. And he said, "Oh yeah, those guys at IBM don't know that there's a border." So what he said was, "You have an appointment with the INS office in Manhattan. You'd better get there, and they'll regularize this." It was not as if this has happened for the first time, you see.

So I was going to be in Manhattan anyway, and I said, "All right, I'd like to see the guys in Time-Life." And they said, "All right, all right. They're ready." And they showed me what they had already decided to do, expecting me to approve it. They didn't want to know what I was going to propose. They just wanted to know that I approved of what they had decided to do, which was essentially what they'd been doing before.

HAIGH: And this is in terms of--

KAHAN: Exception handling. What do you do with floating-point overflow, floating-point underflow, floating-point divide by zero?

HAIGH: All right, these things that you'd been thinking about since you built the monitor implemented in FPTRP.

KAHAN: Yes. And it wasn't as if their hardware couldn't do it because, you see, the guys at Waterloo-- remember I told you that the guys at Waterloo got a 360 mod 75? Well, the guys at Waterloo had been willing to program this stuff on their machine, so I had the program ready to go. But the folks at Time-Life didn't want to look at it.

HAIGH: So in this case, they wouldn't need any change to the hardware, just to the operating systems?

KAHAN: No, just to the floating-point trap handler, not exactly to the operating system. It's just to this one program, which when you do something naughty, you jump into this program. And then it would have been a very straightforward matter. It would've copied almost all the functionality that I have in that document, and which the SHARE committee had thought was a good idea and had wanted to expose in 1964. It wasn't as if they'd said, "Yes, we approve of doing this." They said, "Yes, we approve of discussing this. We think it'd be a good idea, and we should talk about it with a membership," you see. And the guys at Time-Life didn't want to look at it. They'd already decided what they were going to do, and it was dumb. I can go into that if it matters to you.

So I really felt betrayed. I had met some nice guys at Yorktown Heights and at Poughkeepsie. They had not asked me anything about this conversion process, which is why they'd wanted me to be around, in case they needed me. They had not told me why they wanted to study extended precision, so I thought it was just an academic exercise. That was the way I spent the summer of 1967, at IBM.

Of course, I had other things to do and did. I had a really good time in my researches. That was the ellipsoidal error bounds, which we can talk about, if you wish.

HAIGH: Perhaps, is that research that you published? Is there a citation that goes with that?

KAHAN: Partially. Actually, some of it appears in a book by Fred C. Schweppe at MIT. I think the title of the book is *Uncertain Dynamical Systems*, and he has a little bit of it in there. [F. Schweppe, *Uncertain Dynamic Systems*, Prentice Hall, 1973.] It's certainly on my Web page. I've given talks in various places. It must be in the proceedings of one of the Field Institute proceedings, but I can't remember.

HAIGH: Well, if that material is available to people through these other routes, then I think we should probably move on, unless you have any comments about how it fits into your career as a whole.

KAHAN: Well, we'll get to it some point in there because we're going to get interval arithmetic.

HAIGH: Okay.

KAHAN: If you looked at my mindless paper, the one that says mindless.pdf, that has about as good an exposition of the issues as any. And I think from a historical perspective, what would interest you more is why the interval arithmetic community is so zealous about interval arithmetic, the way they construe it, and so resistant to ideas that I would advocate. And this would get us to [Ulrich] Kulisch and [Willard M] Miranker. Kulisch is a real nut, and Miranker is a pest. Miranker is one of the guys I met at IBM Yorktown Heights.

HAIGH: All right, so do you think that's something we should talk about now, or does that come later in the story?

KAHAN: No, it can come later, it's not an important topic.

HAIGH: Now, I feel something that does come at this point in the chronology, which would be some interaction of yours with George Forsythe.

KAHAN: Well, that was 1966. I spent half a year at Stanford. I think that half year at Stanford was precipitated mainly by the paper that Golub and I wrote in 1964. [Golub, G H, and Kahan, W, Calculating the singular values and pseudo-inverse of a matrix, SIAM J. Numer Anal. 2 (1965), 205-224]. This was the paper about computing singular values. We both had about the same idea at about the same time, and we got together and merged our ideas in a paper that subsequently, apparently, became extremely influential. It became influential mainly because we showed that there was an economical way to perform a computation that had previously been inaccurate when done in an economical way. Our economical way was different because it was accurate—about as accurate as people thought it should be in the light of the error analyses of the time. Now, mind you, I have subsequently written—co-written really—papers with Jim Demmel and he and his friends and collaborators have extended this well beyond anything that I suggested initially. So they can get accuracies rather better than what Golub and I would get under circumstances where the better accuracy is justified. For the majority of problems, what

Golub and I did got about as much accuracy as people had hoped to get. And we got it at a very tolerable price. I don't know if I have it here somewhere. Let me see. I think it must be in some other heap. But anyway, there is only one paper that Golub and I collaborated on in 1964, and it appeared in one of the numerical analysis journals. And, as I said, it made quite an interesting splash. I guess you must have it in a heap over there. So that enhanced my credibility in some circles and, of course, I had contributed to SHARE—which enhanced my credibility to some extent with Gio Weiderhold.

Why did that matter? Well, you see Gio Weiderhold had worked at Berkeley until he found himself at odds with the people above him. The people above him were really computer illiterates. I think one of the people above him was the then-vice chancellor for research, who was actually in the agricultural school. It was obvious that he was in over his head. The computing center had lost a number of directors. They had tried to find directors who would carry out the policies enunciated, et cetera, et cetera, but they couldn't find anyone. Inevitably, the computing center directors would have to do other things than the higher-ups thought they wanted, and the computing center was faced with an impossible situation because of a change in federal policy.

I don't know if you're aware of this, but in the mid 1960s, the federal government decreed that no longer would federally funded research pay one rate for computing (a high rate, as it happens) and other users of the computer pay a low rate or nothing. Well, what that meant was that the other users of the computer, frequently for instructional purposes, could not be subsidized. Somehow, funny money or other money would have to be found to pay for instructional computing—but there was no precedent for such an item in the budget. There was no way they could see to insert it into the budget so the state legislature would accept it, so this started to precipitate serious deficits in the computing center and an attempt to find various directors to see if they could dig their way out of a hole. It didn't work because it was a hole that was intrinsic. So what they had was really an acting director who was just a lackey. He was really an office boy. And he fancied himself to be a programmer, but he wasn't. He did not understand a number of software issues, so he and Weiderhold were at odds, because this lackey was friendly with the vice chancellor in charge of research.

These two conspired and found a way to get rid of Weiderhold. You see, they got a request from India, and India thought that the University of California at Berkeley was one of these world-leading institutions, and maybe they should send someone to help the folks in India set up a computing center of their own. And they persuaded Weiderhold to go. "You'll love India. You'll be there for a year or two, and then you'll come back, right?" Ah, but while he was there, his American visa expired, and nobody thought to act to renew it. I wonder why not. So when he came back, he found that he didn't have a work permit anymore. He couldn't work at the University of California anymore.

Well, he got snapped up at Stanford—Stanford Medical Center grabbed him. And it's possible, though I can't say for sure, that if people were wondering whether I should be invited to visit Stanford, if anyone would have asked Weiderhold, then he would have said, "Yes," because he knew from SHARE that I was a kindred spirit. Subsequently, when I did visit Stanford, Weiderhold and I went to visit the guy who had been the chief architect for the 360 line. Why has his name just suddenly dropped out of my memory? There was a company bearing his name that, ultimately, got together with the Japanese.

HAIGH: [Gene] Amdahl?

KAHAN: Amdahl, that's right, Amdahl. We went to visit Amdahl in Sunnyvale. That's where his office was. We wanted to ask him why he had done these things to the 360. You have to understand, this was in 1966, and we hadn't yet gotten IBM to agree to make any changes. And Amdahl acknowledged, "If we had known then what we know now, we wouldn't have done it," a familiar story. So that was nice, but it didn't mean he had the power to change anything.

So Weiderhold and I did cross paths, and Forsythe was sufficiently impressed by some of the stuff that I wrote that he actually wrote a couple of papers of his own on the interesting challenges of doing something so trivial as solving a quadratic equation. He used some of the material I had lectured on. That was not the solving the quadratic equation. It was one of these transcendentally important problems. It was just that computer scientists who had developed a resistance to the ideas of the calculus and the continuum nonetheless knew how to solve a quadratic equation because they'd learned about it in high school. So I was using that as a vehicle to explain why there were certain problems, and Forsythe actually wrote a paper using those ideas. But I gave a course or two at Stanford during that time to explain why numerical software had to be engineered with some extra consideration that was not visible to the people who'd been doing computations by hand, and was definitely not visible to the people who had started to take over architecture.

What is an architect? An architect is a man who designs a building for someone else to build, and yet someone else again to live in. Well, the architects of what came to be called computer architecture, and the architects of programming languages were no longer the people who would have to earn their living by using these things day by day. So they had their own vision of how things should go, in many cases guided by aesthetic rather than quantitative principles, and under no circumstances would they actually look over the shoulders of the people using their machines. And so I had to try and explain that current architectural trends at that time were inimical to numerical processing and that we should do certain other things. So it was an interesting six months at Stanford.

I shared an office with Joe Taub. I think I mentioned to you that Joe thought he would volunteer to be my scientific executor. I don't know whether he still feels that way, probably not. Joe and I are not enemies or anything, but we've drifted apart mainly because Joe was making what I and others, like Beresford Parlett, construed as extravagant claims for his computational complexity—information-based complexity, as he calls it. These extravagant claims—it wasn't obvious that these were extravagant. I know that Dick Karp, for example, didn't see anything wrong with the claims. But Dick should have known because Dick Karp has enunciated a lemma. We call it Dick Karp's lemma. Anything you do, no matter what it is, can be proved optimal if you choose the appropriate criteria. You see, the difficulty comes with the word *optimal*. Joe and Henryk Wozniakowski, who is a very bright guy, a very competent mathematician, but essentially, a mathematician. Joe is somewhat of an operator, and they publish both papers and books proving various things, various numerical things to be optimal. And certainly it was the style of stuff that appealed to people like Steve Smale, so he has a certain audience.

But, in some instances at least, there is a fundamental flaw. Do you want me to digress to that?

HAIGH: Is it something that you've written about anywhere?

KAHAN: No, I haven't. Beresford has, but I haven't. And I could probably articulate the flaw better than Beresford has.

HAIGH: Okay, so articulate the flaw, and then after that, go back and say what Taub was working on in those days.

KAHAN: All right. What we'll do is consider one of their schemes, which is for quadrature. They published a paper, which seemed to create a stir when they said, "Adaptive quadrature cannot be optimal." Remember I told you I'd written an adaptive quadrature program?

HAIGH: Yes.

KAHAN: Okay. I hadn't claimed that it was optimal by any means, but their paper, which said, "Adaptive quadrature *cannot* be optimal," was clearly mistaken. Let's see what the mistake was. We define a class of functions. For them, the class of functions is functions defined on a certain interval. The interval is not important, so let's just say it's from minus one to plus one. What matters is, how big is the *k* th derivative? And suppose you know that the *k* th derivative can't be any bigger than some number *m*. These two parameters *k* and *m* characterize a class of functions. Now, what you would like to do is produce a quadrature procedure—that is, a sampling procedure—which will sample these functions. For a given tolerance, epsilon (so now we have another parameter. *k* is the order of derivative, *m* is the bound on the case derivative, and epsilon is the error you're willing to tolerate) what we'd like to do is find a quadrature procedure which will have an error no bigger than epsilon for every function in that family.

Now, the quadrature procedure will consist of sampling at certain points, multiplying by appropriate weights to get an average (well, actually, you have to divide by two, because averaging from minus one to one is two, which is the interval). Let's talk about the average thing. Integral divided by two—that's what we want to do—so it's really an average. Therefore, the things that are available to you as you look for algorithms are: where should you draw the samples, how many, and what should the weights be. And what they did was to show that there is an optimal distribution of samples and weights, optimal in the sense that it has the minimum number of samples for which you can be sure that the error will finally be less than epsilon for every member of every function in this family. And the distribution is foreordained. And if you use an adaptive quadrature rule, which subdivides and so on, you'll never get this distribution. And therefore, it can't be optimal. That sounds good. I bet you're convinced. So what's wrong?

HAIGH: The amount of work you'd have to do to find out the optimal distribution of samples and weights?

KAHAN: No, well, once you tabulate it once and for all, what are the parameters? They're *k*, *m*, and epsilon. You tabulate the parameters in advance. For any *k*, *m*, epsilon, you look them up, and there they are. So how can this not be optimal? The reason it's not optimal is because we don't know *k* and *m*. When we have a quadrature problem, we know the integral. We know how to compute the function. We may know a lot about it, but we don't normally know a bound *m* for any particular derivative *k* that matters, or if we do, there're a lot of choices. Which derivative? Which choice of *k*? And for each derivative, what *m* do you get? And you know, finding a bound on the derivative can be just as hard as evaluating the integral. So you see, what adaptive quadrature does is it relieves you of the necessity of determining the *k* or the *m*. It's not doing what they said. It's a different problem so, of course, adaptive quadrature, for solving a certain problem-- why should it be optimal for solving some other problem in which you know all sorts of things that in fact in practice we almost never know? Got the picture?

HAIGH: Yes.

KAHAN: Okay. Well, mathematics was clever. The mathematics was Heinrich's. The mathematics was clever, but the sale, the selling job, that was Joe's. That's how Joe and I parted company, and there were a few other instances in matrix computations where the same thing would happen. And Beresford Parlett would write about that. So what was Joe doing? Well, Joe was working with a graduate student, Mike Jenkins, on what became the Taub-Jenkins algorithm.

HAIGH: Yes, he spoke to me about that.

KAHAN: He's spoken about that, and he's proud of that. It's a very complicated algorithm. There is no prospect of proving it is correct in any sense because we know that it fails from time to time. It doesn't fail very often. And in some respects, one could argue of exactly what is a failure because, you see, to find the zeros of a polynomial begs a question, which is that the computational resources available to you, namely the precisions of the coefficients and the precisions of your arithmetic are adequate to determine the zeros in a reasonable way. In fact, as Wilkinson showed in writings that are perfectly transparent, one can take a polynomial whose coefficients look relatively innocent and discover that if you make an error in the 14th or 15th decimal place, you change some of the zeros in leading figures. That's because the zeros cluster in a way that is not obvious to the naked eye. They become hypersensitive to perturbations. And when I say hypersensitive, I mean hypersensitive by factors like 10^{15}. And therefore, it is possible to argue about whether you've computed the zeros correctly when you haven't even got one figured correct for the polynomial you started with.

So we need a somewhat different criterion for deciding correctness there, and the criterion would be, roughly speaking, that the zeros that you allege are the zeros of a polynomial whose coefficients are in some reasonable sense indistinguishable from the coefficients that you started with, or even they're distinguishable only in n figure perturbations. And if you think that's easy, you just don't realize. It's a horrible problem. It's not been solved properly yet. Modulo that problem, allowing for that uncertainty, their program occasionally fails. It occasionally fails to deliver a decent answer even for that problem. The way it fails is it runs too many iterations for something. It gives some sort of unfortunate message. Remember I mentioned Brian Smith and his Laguerre program?

HAIGH: Yes.

KAHAN: That one, so far as I know, has never failed. Furthermore, it's four times as fast. Well, all right, Joe did a good job of selling, and that's a reasonable thing to do. I mean, there's a long tradition in doing that sort of stuff. If you've put a lot of work into a program, and you're proud of it, well, I guess you're going to try and sell it. And I don't have any war with that. I think it would be a good idea to find a student to work on failure modes to explain what they are and why. And I'm not interested in working on the failure modes for his algorithm. I'm more interested in finding a student to write a really good program. Joe is a more political figure than I am, by far. And he had certainly participated in projects at Bell Labs, where the problems of portability had become acute because they had so damn many different computers. And, of course, they wanted to try to cut down on the incidents of reprogramming when you try to solve the same or a similar problem on this computer, for which you have a solution on that one. So they put a lot of work into this. I think Stan Brown, W. S. Brown, was one of the intellectual founders there, and there were a number of people who did a lot of good work on that. But I feel that their library did not have as much impact as it deserved.

HAIGH: When you say that, are you talking about the PORT Library?

KAHAN: Yes, that's correct. I explained to you that there's an NIH factor—not invented here. There's a very, very strong tendency for people to use their own program, which they believe they understand, rather than use a library program, especially if they have to read its documentation. I think the PORT library had a lot of good ideas in it, and although I might not agree with everything that was in there, I regard the work as deserving an accolade. I particularly respect Stan Brown, even though Stan Brown figured in what I regarded as a catastrophe, but with the best intentions. There's something called Stan Brown's model. It's a model for floating-point arithmetic designed to help people write portable software. Now, it fails in this attempt for a reason that wasn't obvious when he printed it. What he does is describe arithmetic rather than prescribe it, and the description says, "Well, when you perform an arithmetic operation, you should get a result no worse than if you had done thus and so."

Because this description can be framed in such a way that it looks like axioms, although he had to come up with some 30 odd axioms, and even then, it still didn't cover the CDC 6000-class machines. He never did find a workable set of axioms that covered all commercially significant schemes, so he had to leave those out. And we can understand that because you've got one of my papers, "A Survey of Error Analysis," which is particularly critical of the CDC-class machines. [A Survey of Error Analysis, W. Kahan, pp. 239, in Information Processing 71 (Proc. IFIP Congress 1971) North Holland, 1972]. So now, you've got axioms; therefore, you should be able to take a program, apply the axioms, and deduce things about the program. And if you can prove that the program is correct in the light of these axioms, there you are. You've got a portable program.

It sounds good, doesn't it? It sounded good to a lot of people. I tried to explain that, no, this was an awful thing. Why was it awful? It was awful because the axioms characterize a computer whose arithmetic is worse than any computer ever built. You see, if you allow arithmetic to do what the axioms permit you to do, then you allow the arithmetic to be more capricious than any existing machine's arithmetic. On existing machines, they may commit some crimes, but they don't commit all crimes, you see. And these axioms, in order to encompass all machines, they allow all crimes. So what they do is make it impossible for a programmer to test his program. He can't test it on the axioms because there is no machine that bad. And since he can't test his program, he doesn't know what he should prove. He has no way to experiment. After all, it isn't just that proof springs full blown from the imagination. You've got to have something to base your proof on, have some idea of what can happen and can't happen. Sometimes the only way to find out what can or can't happen is to look and see. Maybe you won't test everything, but at least you'll get some idea. So I thought this was a disaster. It was a very clever idea and nicely expounded, and so persuasive that it got absorbed, adopted by the programming language ADA. They thought this was just beautiful, and I tried to explain, "No, don't do this! It's a giant backward step!"

van Veinhagen had tried earlier to propound a set of axioms with the same view. His axioms were more restrictive. I think he had only 20 axioms. The axioms for real arithmetic, I'm told, can be condensed to 12—although, really, saying that there are only 12 axioms is saying like the American Revolution had exactly 20 causes. So let's not pursue that too much. Let's just use these rough numbers as an idea of the complexity, in some sense, of the axioms. So if the real numbers have a complexity of 12 axioms, and van Veinhagen's is 20, and Stan Brown's is 32. You can see why Stan's doesn't look like an advance in science. And you see, the alterably standard stuff is prescriptive not descriptive. You can deduce what the arithmetic does. You can deduce which axioms it satisfies or doesn't satisfy, because you know exactly what it's going to do. Therefore, you can prove things about programs. And the great step forward with the IEEE standard arithmetic was, not only did it have a relatively concise mathematical characterization, but you could build it in such a way that it didn't go significantly slower than if you'd been sloppy. It didn't take extraordinary effort to get the arithmetic to work the way the alterably standard says. And once you've done it, you've done it, so now you just have to replicate it. That was the way I felt we had to go, but this does not diminish my admiration for Stan Brown. I think he's a really clever guy.

HAIGH: So returning to your time with Stanford: I know, at that time, George Forsythe was extremely prominent in computer science and in numerical analysis. Could you describe your impressions of him?

KAHAN: Of Forsythe? Well, Forsythe is another one of these Christian gentlemen—very decent guy, trusted by everyone who knew him. They trusted that George would act conscious of the interests of the people around him, and he did. They trusted that he was intelligent and knowledgeable, competent. I don't know of anybody who had anything less than admiration for him, and certainly I admired him. He was very kind to me during my visit there, hospitable. He made arrangements to make available whatever I felt that I needed, not that I needed a hell of a lot. And above all, he had time to chat with me when I needed somebody to chat with, and he encouraged me to continue in the work that I was doing. He felt, as not many other people did, that these were important issues. You see, the prevailing view among most people, including people who use computers a lot—for example, the ones who design bombs—was that floating-point arithmetic is about numbers whose digits start here, and after a while, if you move far enough to the right, they sort of trail off into a fog. And exactly what they do doesn't matter. I've already told you that if you have to deal with uncertainty, it's a good idea to sharpen up your language so you can at least know what you mean when you something. You can at least know what your program does when it does something, and then you can deal with uncertainty in a constructive and analytical way.

But that wasn't the zeitgeist. Forsythe saw this in an early stage and encouraged me. I think that encouragement was extremely important. I was a young professor. I think I was already an associate professor by then, but I don't remember exactly. So I was a young professor. I was venturing into an area where there was a great deal of resistance, and it would have been a hell of a lot easier for me to say, "Oh, the hell with it; let me just work on problems in Hilbert space," as I did with Chan Davis, you see. Why not do that? It's a hell of a lot easier. You can write papers that way and be comfortable, but here I was really battling against the big guys. So yes, it was important. Forsythe's influence was important, and in 1972, I think it was the ACM that gave some sort of Forsythe prize, and I was, I think, the first recipient. And there's a sense in which I treasure that prize more than others more prestigious, in a way, because I remember Forsythe well. I'm grateful for his encouragement and his assistance.

I told you at the beginning, I'm not a self-made man. I've had extraordinarily good luck in encountering people who helped, even if they didn't always like me. I mean, M. V. Wilkes probably didn't like my style, but I don't think I could say that I suffered when I owed him because of that. He was hospitable. Forsythe was hospitable. What can I say? It's extraordinarily good luck. I mean, I'm obviously not all that pleasant a person, so they didn't do it because I was loveable.

Well, at Stanford, I wrote a number of papers on various things in order to demonstrate that arithmetic was stranger than fiction. I didn't see Gene Golub as much as I might have liked. We didn't specifically collaborate on anything. I think he may have been away for a while. But something did happen that was going to have a major effect on my life, though I didn't know it at the time.

What happened was a United Airline strike. I had arranged that my family would fly back to Toronto on United Airlines flight out of Reno, because we were going to take a little vacation in the Sierras. And after that vacation, they would take the United flight from Reno, and I would drive our Volkswagen back across the continent. And we went up to Reno and gave United Airlines our luggage, the parts that we wanted to take back to Toronto and weren't going to need for our vacation, and they accepted it—though they knew at the time that a strike might occur. They didn't mention that to us. It wasn't until a day or two or maybe three before I was expecting to fly my family out of Reno. I was going to drive them up, and so on. It wasn't until then that we read in the paper that there was going to be a strike, so we had to go up to Reno and get our luggage. That made it very uncomfortable because there were two adults and two small boys and a certain amount of luggage all packed into a small Volkswagen Beetle. You may not know, but if you look at Volkswagen, the old Beetles, there isn't a hell of a lot of space in those. I had to be an expert at close packing, Schliefly integrals notwithstanding. So what were we to do? The house that we had rented was no longer accessible to us, so we had to stay at a hotel and wait for about a week to take an American Airlines flight out, because that was the soonest we could go. That flight was going to fly out of San Francisco. So I had a week to spend, and I spent that week some of it writing some more reports for the Stanford series on various interesting things that interested me. And I spent a few days with my family. We went down to visit Monterey, and then to visit Point Lobos. Have you ever been to Point Lobos?

HAIGH: No.

KAHAN: Well, if you come out to California, you should see the aquarium in Monterey. You should look at some of the houses in Pacifica, and as you head further south, maybe 15 miles, something of that order, you get to a very small state park called Point Lobos State Park. Now, you pay your little fee. Maybe it's ten bucks. You drive on and then head south. Go to the southernmost parking lot. You must not do this on a weekend because you may not be able to get in or park. So you want to do it on a weekday. You should bring picnic stuff with you because there are picnic tables there. Then you walk a little bit farther south on a trail, avoiding the poison oak. You walk some rickety steps, and you end up on a tiny beach, which is approximately as wide as this living room, and it's at the end of a miniature fjord. And if you wade in the water, just a little bit off to your left, you see a cave; you can walk into the cave. And if you have children, you take them by the hand, and if the tide is out, they can easily inspect the cave. And if the tide is in, you'll have to carry them. It's a spectacularly beautiful spot without being overwhelming. I mean, it's not like the Sierra where the mountains tower way above you. It's just a miniature fjord. It's got this little white sandy beach, this tiny white beach, and if you go and it's not a weekend, you may very well have it to yourself. And we did.

And as I drove my kids back, my younger son Simon, who was two at the time, put the question very simply. He said, "Daddy, why can't we live in California?" Yes, well that was 1966. It planted a seed that sprouted in 1968, because that's when I got the invitation to come to Berkeley.

HAIGH: Well, it seems that takes care of everything that has to do with Toronto, so perhaps we should go now to Berkeley, then.

KAHAN: Well, okay. Toronto treated me extremely well. I haven't got any complaints. They promoted me to full professor even though I said, "I wish you wouldn't."

HAIGH: And why did you say that?

KAHAN: I didn't want to be on these committees. You see, once you're a full professor, you really do have to decide on the promotion of damn near everybody, and it's not a decision that I enjoy making. It wouldn't have made a hell of a lot of difference to my salary, just a small increment, so I said, "You know, why don't we just leave it?" The response was, "Oh, Velvo… grow up." Of course, they were quite right. I mean, I really should take on some of these responsibilities.

But I worked on this stuff to figure out how to compute the uncertainty propagated forward in a trajectory computed by solving differential equations. That was the work I was doing with the aid of Gerry Gable's differential equation solver—a work which has never been fully published because the proofs are terribly long. But what it comes down to that a computer scientist would appreciate is this: all the other schemes for estimating how uncertainty propagates along a trajectory, because where you start is uncertain, and your differential equation, and the laws of motion are a little bit uncertain. And assuming that the uncertainty is extremely tiny—so you can ignore second order effects for the most part—even assuming that, all the schemes published so far have uncertainty bounds that grow exponentially faster than the uncertainty can in the general cases. There are some special cases where they don't. In general, you get an exponentially superfluous growth. And if you look at my "mindless" paper, it says a little bit about why, just a little. It has to do with corners. When you use interval arithmetic, interval arithmetic used naively discusses coffins. These are boxes whose edges are parallel to quartered axes. Interval arithmetic used a little bit less naively considers parallel pipets, which are essentially linear maps of coffins. My scheme uses ellipsoids, which are linear maps of spheres. They don't have corners. It turns out that, in consequence, instead of getting an exponentially superfluous growth, the growth is only proportional to the square root of time. You know, there's a lot of difference between the square root and exponential, isn't there? Well, this is what I really wanted to work on more than any other single thing.

[Tape 7, Side B]

HAIGH: All right, so you've already discussed the kind of big picture situation with Canada that was another element in your decision to go to Berkeley.

KAHAN: That's right.

HAIGH: So had you been receiving unsolicited offers to move to different places?

KAHAN: Oh, yes. I ignored them.

HAIGH: This had been going on for several years?

KAHAN: I didn't keep count, but I ignored them. I had other fish to fry. What can I say? I really didn't want to endure the expense of moving, and I was well treated at Toronto. There's no other way to get around it. Toronto knows how to make people happy. I can describe a case later, and will—Steve Cook, another Turing Award winner. What happened was that one of my friends, Beresford Parlett, whom I had met in New York in 1964; I don't remember why. He was named after, I believe, Admiral Beresford, but I'm not 100 percent sure of that. He had gotten to Berkeley in 1964, part of a board of attempts to set up computer science departments, initially by Abe Taub, who was there in 1964. Taub was the guy who brought Parlett in. Lotfi Zadeh had already started to set up computer science activities in electrical engineering. He was notorious in my mind, famous for fuzzy logic and fuzzy sets. In fact, his office is a shrine, people who come from all over the world to pay their respects to the shrine. And I think fuzzy logic is foolish, but he's still a friend of mine, though we differ about this.

HAIGH: We have a rice cooker. I think the Japanese like these fancy things and will pay more for one that says it uses fuzzy logic.

KAHAN: Why would a rice cooker use fuzzy logic?

HAIGH: I'm not sure, but it's written on the front in big letters. It's a feature you pay more for to decide if the rice is cooked.

KAHAN: Right, rice is cooked. I see, okay. Well, maybe next day we can discuss something about that because I once offered Lotfi a challenge. The challenge was to design a furnace controller for a continuously modulated gas furnace in the northeast of the country to keep the temperature comfortable in a brick house.[7]

---

[7] As it happens, I had designed such a thing because that was what we needed in our house. Our son had just been born. That was 1964. We brought him home, and as soon as the weather got cold enough a few weeks later—he was born in August—and in September the weather got cold enough to require some heat. The heat came on, and the furnace flooded because the boiler was made of cast iron and was intended for a coal fire furnace. It had been converted to oil and, whereas coal is steady heat, oil comes on with a rush and then goes off. So the thermal cycling had caused the boiler to crystallize. And crystals are more vulnerable to rust because once a big crystal starts to lose the battle with oxygen and rust, the whole crystal goes with a bang. It flooded the furnace, and we needed something really quick, and I purchased an industrial furnace with a continuously modulated gas flame because we had cast-iron pipes and cast-iron radiators and things like that. And I thought, "My God, once this crystallization starts to spread and become active elsewhere, what's going to happen?" So I thought, "No more of this rush-on and rush-off stuff. It'll have to be continuously modulated gasoline." And it wasn't terribly expensive. These are industrial furnaces. But how is it controlled? It's controlled by two mercury capillary tubes. One of them is supposed to be wrapped around the boiler, so if the boiler gets really hot, the mercury will expand and definitely shut the gas down. And the other is supposed to be connected to whatever it is for the temperature that you want to control. And as that thing gets warmer, it'll expand the mercury, and there'll be less gas. It was not at all like an electric thermostat. How was I going to control that?

And I figured out a way. I got to read the appropriate handbooks, and we had tenants upstairs who had been complaining about the heat anyway. So I calculated the size of a sheet of aluminum to bend outside the basement window on the north exposure of the house, exposed then to prevailing winds. And I took this capillary tube and wrapped a measured length of it around that aluminum outside the window, just outside the window. And I calculated it all very, very carefully and closely and repeatedly and did little simulations and so on. If the wind blew, that would carry the heat otherwise escaping from the basement through the window, would carry the heat away, so if the wind blew, the aluminum would get cooler. And that would, of course, make the furnace come up a little bit more. That would compensate for wind. I would compensate also for the fact that when the outside gets very cold in a brick house whose insulation could not be described as perfect. You lose heat by radiation, from the surface of your body to the walls, regardless of the air temperature, so you can think it's colder than it really is. So I said, "That means I have to increase the temperature a little bit, not to keep the air temperature constant but rather to keep people feeling that it was about right." So I read handbooks and handbooks to describe these things, and I calculated it all.

And I set it all up, and it all had to be done in a hurry. It worked perfectly. The upstairs tenants adjusted the thermostat, which, of course, had been disconnected, but they didn't know that. And they were perfectly

And I challenged Lotfi Zadeh. I said, “Let’s see if you can use one of the fuzzy temperature controllers used by Fujitsu or anyone else. I don’t care. Design a controller.” I said, “It’s going to cost you a hundred times as much to build it with that fuzzy controller and at least ten times as much to design it with that fuzzy controller. Want to try and see?” He never took me up on that. I don’t think he never will. It’s not really what he wants to concern himself with.

But Lotfi Zadeh, back in the early ’60s, could see the writing on the wall. The electrical engineering departments could not survive if they were going to stick with the conventional electrical engineering. They were going to have to move into the computer world, so he set up a sort of clandestine little computer science department in there. He hired some people, and some were very good and some not so good. Taub was doing the same thing clandestinely, because he didn’t really have a license to do that, although the math department had brought him in specifically to do that. But he didn’t have a license—that is, from the dean—in order to do that.

And then, finally, some of the members of Zadeh’s computer science group became dissatisfied with the way in which he hired people. For example, he decided to hire some guy whose name I would rather not mention—a guy who was prominent in IEEE activities, prominent in organizing conferences and stuff like that. It was a guy about whom Lotfi had gotten glowing letters of recommendation from the people at the university where this guy was. And I won’t mention that because then you’ll really know who he is. So Lotfi hired him away and brought him to Berkeley. That man was a blithering idiot. In fact, ultimately, he was the butt of jokes around our place. You see, the letters of recommendation had been cooked. He’d received these enthusiastic letters of recommendation from the people who wanted to get rid of this guy, and Lotfi had fallen for it hook, line, and sinker. He hadn’t gone through the appropriate procedures of review and so on. He’d bypassed them because he was the executive type. So some members of his department were chaffing under this, that this guy was building up a little empire and was not getting the right sort of folks, not even trying to get advice from competent people, so they wanted to set up their own little computer science department.

They got a license from the dean of letters and science to set up a computer science department within letters and science instead of engineering. They had recruited some really good people, like Dick Karp and some fairly good younger guys, and I wanted to help Beresford. I thought, “Why don’t I go down there. I’ll work down there for seven or eight years, and then we’ll see what else we’ll do.” So that’s why we moved. Part of it was because my students could hardly get jobs in Canada. They were going to go to the U.S. anyway. Furthermore, Canadian graduate students could often do their graduate work in the United States, often did. I mean, at Stanford I had met a few of them. So why not? But the thought of helping to set up another computer science department and work with some really good guys, for at least a while, attracted me.

HAIGH: All right, so at the time you were hired, did you have the same arrangement you had at Toronto with the joint appointment between this--

KAHAN: Actually, I had a triple appointment. I was appointed jointly in mathematics, in this new computer science department, and in the computing center. My hope was, and their hope was, that I would raise the tone of the place. I wanted to use my experience with SHARE and

---

comfortable, no more complaints. And of course downstairs, where we lived, it was fine. The only thing you’d have to adjust were the valves on the radiator so that radiators in different rooms would radiate more heat or less depending upon how rapidly circulation occurred.

with Toronto and so on to enhance, very substantially, the quality of service that people would get from a computing center, not administratively but rather scientifically. And there were some people there that thought it looked as if I could build on that. But I didn't figure on the man who was the acting head of the computing center, who was an idiot.

HAIGH: That would be where this committee report comes in?

KAHAN: I was on this committee starting-- I arrived in January of 1969, and I don't know how many weeks it was before I found myself on this committee. Well, I thought we had done a conscientious and good job. People who were on that committee included guys from the chemistry department and from other interested departments, all of us academics, and our recommendations, we thought, welcomed by the administration. But the only recommendations that were adopted were the recommendations we made to terminate the sinecures that Taub had established. Remember he wanted to set up a clandestine computer science department. He did it by hiring people, ostensibly, for the computing center. One of them was Bob Baer, who was essentially a logician. He had Rene DeVogelaere. Rene was a mathematician who got into computing early. He died some years ago. He had been on the math department and then got this half-time position in the computing center. Martin Graham had been part of the electrical engineering department, and had actually participated in the building of a computer at Rice University. It was a goofy computer, but he participated in the building of it. He had some experience there and was also very expert in communications, generally all around a very good electrical engineer. He had a sinecure, and I had one. And maybe Parlett had one, too. I don't remember. But Parlett had become chairman of the new department. So whether he had a sinecure in the computing center or not didn't matter. He had a full-time job as chairman.

And these positions had been established by Taub in the expectation that if a computer science department was formed, it would absorb these people into the department. But the new department—the guys who'd formed the new department—they didn't want Rene DeVogelaere and for good reasons. He was a very difficult person, a difficult personality. They didn't want Bob Baer. Bob Baer was competent, but he wasn't up to the academic level of the other people that they appointed. They had Marty Graham. He was already a part of this little group, and they had me. And Bob Baer felt a little bit miffed, and DeVogelaere felt very miffed.

In the meantime, the computing center was suffering a hemorrhage that seemed to be a constant of nature: $150,000 a year, no matter what they did. So our committee met and deliberated, and first we said, "You've got to get a competent director." They'd gone through several directors, you see. That's why they only had an acting director at the time. And they had a director—Len Stewart, I think, was a director for a while. He was a good guy, but he wasn't a good director. He just simply made the budget deficit get worse and so on. Everybody would promise to eliminate the budget deficit, but they couldn't. You see, the budget deficit was built in because of this change in federal policy and the refusal on the part of the state legislature to introduce a line item in the budget for academic computing. Well, they got there finally—but only after an enormous political battle. But anyway, so we made our recommendations, and the administration ignored all of them except the disillusion of the sinecures.

Well, Bob Baer didn't like me after that. Rene DeVogelaere wasn't altogether fond of me, either. Martin Graham was okay, because he was actually involved in setting up the communications for the networking that was to come. I was happy to abandon the sinecure; that was part of what I had recommended. Let's try to get rid of this budget deficit. I had appointments in two departments are enough, really. And part of it was chairman, so he didn't matter either. But when

I saw what the administration did, they promoted this lackey and made him head of the computing center. And they disregarded all of our other things except the elimination of the sinecures, while saying, "Thank you. We're going to do this." Well, I decided it looked as if committees were futile and diversional; I'd do my damnedest to keep away from them. I had done that for 36 years, except for the committees who had won.

So I established a practice of reviewing the progress of graduate students in the computer science department every semester. That was established, if my memory serves me rightly, somewhere around 1975 or thereabouts, maybe a bit later. We lost two faculty members, one of whom went to a southern California university where they would pay him more. God knows he needed it, because his wife had socked it to him during divorce proceedings. She'd found a lawyer who was extremely effective at squeezing this poor guy dry. As you gather, it must have an extremely acrimonious divorce. It was. And we lost the other guy because he did not heel to the current fad, which was called *RISC*, Reduced Instruction Set Computer architecture. It—what can I say?—it was a fad. Its virtues were oversold. Now you look at RISC computers, and they look just as complicated as the complicated instruction set computer. So what the hell? But that's a story for another day.

So we lost two faculty members, and some of their students were left to cut adrift. What're they going to do to finish their theses? So I established this review process. Part of the responsibility of the review process was to make sure that every student was mated with an appropriate thesis advisor, one way or another, and also to make sure that faculty knew that whatever their relationship with the student, it was open to scrutiny. We want not to interfere. I mean, interfering in that relationship is a generally bad idea, and you want them to interfere as little as possible. And the way you interfere as little possible is to intervene as early as possible to make a small tweak. It's like having lunch with somebody and saying, "How are you doing? Are you having trouble with this sort of thing? Do you have plans for dealing with it?" You know, quiet little conversation, and then nothing has to go into a file. It doesn't have to be discussed in a faculty meeting. These things get resolved simply because folks know that somebody's looking. We've jumped ahead. I'm sorry.[8]

[8] HAIGH: I saw a reference on your Web site. It was a URL to computer science ombudsman.

KAHAN: That's right.

HAIGH: Is it the same thing?

KAHAN: No. Actually, I don't know if I've been renewed as ombudsman, but I probably have. I was computer science ombudsman only because my colleagues think that I'm less impatient than they are. And the only reason there's any truth in that is because my wife was trained as a social worker, and she expects a certain level of behavior. And I wish not to disappoint her. I'm fond of her, and I don't want her to think that I will disregard things that are important to her. So I listen to people. I talk a lot, as you've gathered, but if someone comes, I will listen. I will try to understand their point of view. I'll try to figure out what can be done without-- An ombudsperson has no power doing it. I can't coerce anybody. I can't give anybody orders. Of course, some of them don't know that. So, for example, on one occasion, a young undergraduate female came to complain that a teaching assistant of foreign extraction had "dissed" her—had said that girls make terrible programmers, so there's really no point in his investing any time explaining something to her. He was making other comments about the inadequacy of womankind in the computing profession, so she complained about it. She complained first to the professor, and then the professor said, "I don't know what I can do about it. You'd better go to the ombudsman." And I listened to her, and I checked with the professor to find out the professor didn't know because the professor didn't go to the sessions

Okay, so I arrived in Berkeley, and I was on this horrible committee. And I will not get onto a committee like that again. And I acquired a graduate student—that's Betty, whom I inherited from Beresford.

And the story of Betty is a bitter story, and I want to start with the story of Em at Toronto, because it's part of an unfortunate pattern that we don't properly appreciate. Em was a programming assistant. She needed a job. Although she had stood first in mathematics as an undergraduate, she wanted a part-time job at the university. She did not feel like continuing as a student, so she got a job in the computing center and was assigned to work for me. I gave her work of ever-increasing complexity, and she dispatched it all in an exemplary fashion. And when I looked at her work, I realized that I was dealing with somebody who was really very, very bright. Her husband, upon graduation, had gotten a job in a government location as a technical expert. So he was making pretty good money. Em, however, had recently given birth to a small child, and she was working only part time, just to have something to do.

One day, she said she felt that she should spend more time with her child now because the child was old enough that just putting her in a crib under a watchful eye was no longer a satisfactory way to manage. And I said that childcare is available. I know because, at the time, we were taking advantage of the childcare system at the university where, while studying the behavior of children, they watched over them. And that's where our younger son went. And it was educational for him, and it was educational for them. There is this sort of thing available, but I told her, "You would be an adornment in any graduate department I can think of. I'm confident that you could get a Ph.D. and then, in due course, instead of my telling you what to do, you'd be telling me what to do."

But she didn't want to do that, and the reason she gave was that this would outshine her husband. He was known to have a somewhat delicate ego, and she had no intention whatever of bruising it. And therefore, she was going to enjoy time with her child. So the promise of a career, a promising career, was something that she wasn't going to pursue, at least at that time. And she never did, to my knowledge. I was very saddened by it. I understood her arguments. I understood why she did it. I was saddened mainly because, if her husband had had a less unenlightened attitude, then I would not have to have been witness to what I had to regard as a waste of incredible talent. She could have spent time with her child. The child would have been in

---

over which the teaching assistant was residing. So I had some quiet conversations with fellow students and found out that her complaint was quite justified.

And then I had a meeting with this teaching assistant. Now, he didn't know that I didn't have the power to fire him, and I exploited that misapprehension on his part to tell him that, first of all, the things that he was saying were not true. And I could cite chapter and verse. I've already mentioned Beatrice Worsley, Cecily Poppelwell, and there were others that I could cite. So I told him, aside from the fact that he was wrong and that, therefore, he was promulgating falsehoods, he was also insulting people gratuitously. And if he wanted to regain my respect, he would have to overcome the cultural bias of his country of origin and apologize to this young woman in the same context in which he had insulted her, which means in one of the TA sessions. That was the only way, I said, that he could set things right insofar as they can be set right, because there was a certain amount of damage done. In any event, I wanted him to know that he was very much mistaken. Well, in trembling and in fear, he went and did that.

I don't know whether I changed his attitude towards women, but I've certainly changed his behavior. That's an extreme example. Normally I would not be able to exert that kind of authority and get away with it. But I think there is a phrase that I've read somewhere: "You can do anything if you do it only once." I can't remember where I read that, but anyhow. Okay, so that's-- I've been ombudsperson for about 20 years.

childcare not very far away from where we were working. She could have worked only part time. A half-time graduate student may not seem like much of a graduate student, but she was so bright. She could do in half time what somebody else would have to spend extra weeks on. So I was sad that the opportunity was lost.

Now, we come to Betty. Betty had been a graduate student for 13 years when I arrived. Now, that's a rather long time. She had been a graduate student in a chemistry department across the street, where she had met the man who was soon to become her husband. But she had changed to math from chemistry very recently, just a year before I arrived, and Beresford was her thesis advisor. And Beresford had been unable to find a topic that ignited her enthusiasm. She would take courses and ace them. She would solve all the problems in assignments beautifully and with a flare of ingenuity. Beresford thought that maybe I could find something that would suit her. So the first thing I wanted her to do was to translate Jerry Gable's program from the 7094 in Fortran to the CDC 6400 in Fortran so that we could then use it for research.

HAIGH: And the 6400 was the main machine in the Berkeley computer center?

KAHAN: Actually, they had two of them, and they were using at least one of them to develop a time-sharing system. Butler Lampson was the academic. He was a member of this little department, and he was the academic who was involved. Gio Wiederhold was gone. There were a couple of his minions who were still working with that. Nothing much ever came of it, but I don't think it was altogether their fault. I think they were fighting a losing battle against the rapid obsolescence of a mainframe idea. After all, this was 1969, 1970. The minicomputers came in, and if you could afford a minicomputer—the service bureau group—why should you become involved in a mainframe servicing a much larger number of people? In fact, the minicomputers were, at that point, almost cheap enough that, with a couple of grants of the right kind, you could afford to have your own computing center with all the responsibilities and aches that that implies. But of course, they wouldn't know that at first. First you've got to get sucked in, and then you realize, "God, I'm over my head." Well, okay, so Betty transcribed the program, and she encountered difficulties. There were examples that didn't seem to work right, and she would come to me repeatedly with output.

And I could look at the output. I could look at the listing of the program, and it was perfectly clear that that program could not possibly produce this output. It was just not possible. What the hell was going on? And we worked at this for a while. Her program was just fine. I mean, I looked over her program repeatedly. I really didn't see what could possibly be wrong with it, but I saw that she was showing the signs of a nervous breakdown. And I thought, I've got to get her off this. So I switched her to a theoretical problem, also in the field of differential equation. It was essentially to prove that this kind of program deserved to work. A Polish author (whose name I have quite forgotten) had proved something like this, but under circumstances that were more restrictive. And his theorem was wrong. I mean, it looked to me as if it was a mistake. And she found the mistake. So once she found the mistake, I realized she understands this stuff. Now, let's see if you can do it right. It doesn't need that restriction. So she started working on a thesis.

In the meantime, I really got worried about why the hell this program doesn't work. I thought, The arithmetic must be misbehaving. Well, you know, the IBM machines are tempted to misbehave. Maybe the CDC arithmetic misbehaves, too. So I wanted to look at the circuit diagram. Of course, I wasn't supposed to look at the circuit diagram. That's all proprietary. Nonetheless, I became friendly with one of the service engineers and, one day, I discovered the manuals for the CDC 6600—the CDC 6600 and 6400 were very similar. I found the manuals for the CDC 6600, including circuit diagrams in my wastebasket. The new manuals had arrived, and the service engineer had decided to throw out the old ones. And he threw them into my wastebasket. Aha! Now I could look and see what was going on, and I realized what was happening. I could look at the circuits and figure out that it doesn't have a guard digit on subtract. And what the hell is this going on with this exponent logic? That's weird, and so on.

Then I had to find a graduate student in computer science who liked assembly language programming, and there are people who have this perverse taste. I think his name was Lindsay. It may have been Bruce Lindsay, but I can't be quite sure. I know his last name was Lindsay. So far as I know, after he got his graduate degree, he went off to work for the National Security Agency and, for all I know, he is still there. I mean, he just disappeared from view. And if he's working for the National Security Agency, they've got a really good guy. He's very clever. So I described for him the kinds of things I feared must be happening, but I couldn't be sure that the Fortran compiler would do the right thing. It seemed to on my experiments. I said, "Look, let's write some assembly language programs, so we know exactly what we're doing and find out what the hell is going on." And he confirmed my suspicions about the arithmetic, and he found other things.

He found out what had screwed up Betty's program. I would never have imagined it. He found out that, when the compiler compiled comparisons of double-precision variables, if the double-precision used two floating point words in which the second had an exponent that was 48 less than the leading one, and so on, it would sort of be head-to-tail. He said that, when comparing two double-precision floating-point numbers, if the two heads matched, then, the comparison depends on the tails because of a bug in the compiled code. The comparison can go either way. It appears to be random. You can't tell which way it's going to go. That's what had screwed up the program. You see, we had designed the program to make sure that it would never ever overshoot an endpoint. Because for all you knew, as you solve the differential equation, you come to the end. One step beyond that, and you may be in a singularity, and you don't want to do that. So that meant we used a little bit of double-precision arithmetic to check to see how big a step you could tolerate so you wouldn't overshoot the end; we were using single-precision variables for the step locations, but using double-precision arithmetic to make sure that each single-precision increment was going to do the right thing. And that's why the comparisons had been malfunctioning. At last, I could understand what was going on, and I could set Betty's heart to rest and tell her, "You were betrayed by the compiler."

In the meantime, Lindsay was finding all sorts of other things that I hadn't expected, and he ended up writing a report on what the hardware did, chapter and verse. There were the examples; you could see that this thing was misbehaving, so I took that report to CDC. And I said, "You know, you really should fix this." And the folks at CDC had already heard about my interactions with IBM, and they felt they didn't know what sort of voodoo I could spring on them. So they took me out to an extraordinarily expensive restaurant in San Francisco. This restaurant had been, I believe, a small opera house. And over this very expensive meal, they told me that, regrettably, CDC was not going to change anything.

Meanwhile, a copy of Lindsay's report had somehow found its way to Zurich. I have no idea how it got there, not the slightest. And there came to the attention of Niklaus Wirth. Niklaus Wirth is the guy who was known for Pascal, among a few other things, and he was designing Pascal at the time. He, too, had a CDC 6400, and he had tried to define what floating point should do in order to accord to the principles of his language, but the machine wouldn't do it.

There were really strange anomalies. And at last, with this report, he understood what was screwing up the works. So he wrote a letter to CDC, in which protested that this arithmetic was unreasonable. And I wish I could find a copy of the letter he got. He sent me his correspondence, and the guys at CDC and Zurich said, "Item one: our computer arithmetic does not do this." He's got the evidence, but they say, "It doesn't do this. And second, if it did, it wouldn't matter." Okay? As I told you, the prevailing view about computer arithmetic was you got a certain string of digits, and as they go further to the right, the digits get fuzzier and fuzzier until they sort of disappear into a fog. That's the way they thought about arithmetic. That's the way Seymour Cray thought about arithmetic. That's the way his major customers thought about arithmetic. And do you think the guys at Livermore cared? Of course not, and they're the ones who were paying. So he built the arithmetic that would suit his customers, and it was perverse. And I wrote this thing on "A Survey of Error Analysis" in 1971, when it appeared that there was really nothing else I could do. [A Survey of Error Analysis, W. Kahan, pp. 239, in Information Processing 71 (Proc. IFIP Congress 1971) North Holland, 1972].

HAIGH: And that was presented at IFIP meeting?

KAHAN: At the IFIP Congress in Ljubljana. Actually, I had a certain amount of divine help. I was showing them what happens in various machines, and I said, "Here's what happened with the CDC machine," and this was in a huge hall. It was raining outside. And I said, "Now, let's look at IBM machines," and there was a crack of lightning outside and a peal of thunder—boom-boom-boom! And of course, the audience erupted in laughter, and I had to say, "You know, these technical assistants here are really great." But yes, I was presenting it to people who were predominantly management rather than technical people. There was nothing that I or Klaus Wirth could do that was going to change the CDC computers.

Attempts were made by others to rewrite the compiler. In fact, I think, at the University of Minnesota--

HAIGH: The Twin Cities campus, probably?

KAHAN: Could be. It could be Minneapolis-St. Paul. Somebody there had a CDC machine, and they got a copy of the report. And they attempted to attenuate some of the difficulties. If, for example, for a floating-point add-subtract, you compiled five instructions instead of two, you could alleviate some of the difficulties. You couldn't do a hell of a lot for a multiply and divide, but at least you could change some of the ways in which the compiler tested numbers. So although their compiler generated code that ran slower, it got rid of a certain fraction of the anomalies. It couldn't get rid of them all. It just wasn't practical, but it attenuated them. But of course, nobody used that compiler. It was slower—five instructions instead of two. So all right, the CDC machine disappeared. I don't remember exactly when, but it must have disappeared by 1978 because, by then, I know we had DEC VAXes.

But for Betty, what was the problem? She managed to find what was the correct result, and she managed to find a proof for it. And the proof was pretty good. It was well-crafted, and she was writing it up. And her English was pretty good despite that she was actually Chinese—from Taiwan—from an old family that had suffered somewhat when the newcomers came across with Chiang Kai-Shek and the Kuo-Min Tang. Chiang Kai-Shek killed quite a few of the folks in Taiwan. Not everybody knows that. Betty's family didn't suffer many losses—just enough to teach them a lesson. Betty was very reticent about that because they never knew who might report back to Kuomintang and make like miserable for her family back in Taiwan. So it took a while before I could gain her confidence sufficiently that she knew I wouldn't report her to Kuomintang. But she was writing a superb thesis. In the meantime, she got married to this chemist that she had met. He had a Ph.D. in chemistry, and he got a job with a very important company in San Jose. It was a good job, and things looked up for Betty, as she still could stay at the room in a rooming house where she had stayed before she got married. So she would stay there during the week and then down for the weekend, she'd go back down to San Jose to be with her husband.

And progress on her thesis was exemplary, but suddenly she stopped before writing chapter six. There were only two chapters to go. Chapter six was sort of a coda to the proof, and chapter seven was, you know, conclusions and stuff. That was all that had to be done, but she stopped. "Betty, why don't you want to continue?" "Oh, my English just isn't good enough," she said. "Come on: your English is better than that of most Americans' native-born." And I said, "I don't see any rules that say you've got to write your thesis in English. If you'd rather write it in Chinese, go right ahead. I'm sure we can manage it that way and have an accompanying translation on facing pages." All that amused her, and she wrote chapter six. And then she stopped, and she said to me, "I can't finish it. I won't finish it, and I don't want you to finish it for me." That was it. She forbade me to do anything more with her thesis, and it sits in a file folder in my desk in the math department. And I'm smoldering over it, but it wasn't as if I gave up without a fight. I had asked graduate students to inquire about her results, because they were really rather good. And some of them were interested, and they did ask her. It was clear that they wanted to know about what she had done, and she was a bit shy, maidenly modesty, all very proper. But she did explain some of the stuff.

I even got Hans Stetter, whose name has appeared in your things-- in Vienna, he was the doyenne of the field. I mean, his was the book on numerical solutions of differential equations, and he's a good guy. He's also interested in software and things like that, so I'm sure that you have some interest in him. And I had met him, and I think it's fair to say that we were friendly. And he wrote letters at my instigation asking her about her results, and those were honest letters. He was really interested, I mean, because this was good stuff. She was not going to finish her thesis.

Some years later, I had some consulting work to do at this very prominent company in San Jose, and since I was going to be in the neighborhood, I let Betty know that, perhaps, I could take her and her husband out to supper and find out how they're doing. "No, no," she said. "You really must come over. We'll have you over for supper at our house." So I came to where they lived, and as she was clearing away some of the dishes and going to get dessert from the kitchen-- what I hadn't told you, before giving up, I had gone to the head of the student counseling center who, as it happened was Chinese. And I'd described the problem to him and asked for his advice, and all he could say was, "You've done a great deal more than it would ever have occurred to me to do, and if that doesn't work, I don't know what to tell you. I just can't understand, myself, why she has stopped." So it was maybe four or five years afterward. I came to supper at their house, and she was clearing the dishes and going to get the dessert. And I asked her husband, "How would you feel if Betty finished her Ph.D. thesis and got her Ph.D. degree?" And he looked at me and said, "Sure. Then she could go to work, and I'd stay home." She was just coming in, and she heard that. And a certain dirty look passed between them, and suddenly an awareness descended upon me as to what was happening. And I consulted various references and friends and others. These two came from very traditional Chinese families in Taiwan, and the Chinese tradition, I discovered, was that the eldest woman in the house is the head of the household. It may be the mother or mother-in-law, whatever, but the eldest woman in the house is the head of the household—under the roof. But outside, the wife follows the husband at three paces. He is the one who is the master, so to speak, and Betty had figured that her Ph.D. would trump his Ph.D. in chemistry. And she wasn't going to do it.

At last I understood. I didn't like it. There's something that rose in my throat at the thought, but at last I understood. So I told you about these two women who had their reasons. I might not like their reasons, but that's a choice for them to make, not for me to make. I wish I had understood this better at the outset. I might have planned a different arrangement. I might have planned for her to publish a paper and then write a thesis. And then if she didn't want to write a thesis, well, that would be okay. The scientific work would get out. She would have a reputation based on the paper that would be worth more than a Ph.D. if she wanted to apply for certain kinds of jobs afterwards. You never know if you might have to or not because mean time between marriage and divorce in California is two and a half years. But I hadn't understood her situation well enough. We even went to her wedding, and the penny just hadn't dropped. I hadn't realized what the traditions were and how strong a hold these traditions had upon this couple. Well, okay. We finished the story of Betty, and we finished the story of my horrible committee.

And there's this paper that I wrote. It came out in 1971, and I was a professor teaching courses. And something strange happened that got be involved with Hewlett-Packard, and that might be a good place to begin. That would take us to 1974.

HAIGH: Okay, so we'll pick up tomorrow with a general question on your continued involvement.

[Tape 8, Side A]

Session 5 begins on the morning of Sunday, August 7, 2005.

HAIGH: I know that you are ready now to begin to talk about your consulting.

KAHAN: Well, it wasn't consulting. It was a visit to IBM in Yorktown Heights for the academic year 1972–73.

HAIGH: So before we move on to those specifics, I just wonder if I can ask a general question. In the other interviews, it's become apparent that during the early 1970s with things like the development of SIGNUM, the starting of the *TOMS* journal, and the series of mathematical software conferences organized by John Rice, that there was a sense that a community of people concerned with mathematical software was coming together.

KAHAN: That's correct.

HAIGH: Now, I wonder if you could talk briefly about your involvement in those kinds of general things.

KAHAN: Well, my general aversion to committees kept me away from organizational activities. I was sympathetic, and I know that I must have made some contributions there. I know that I refereed some papers, but I can't remember which they were. And there weren't all that many of them, because it took me a long time to referee something. And I attended some of the conferences. I may even have presented a paper or two at some of the conferences, but I can't remember the details. I know there was one organized by John Rice. I'm pretty sure I attended one of those. I also visited Purdue on at least on occasion, but I can't remember what it was about anymore. Cowell and company at Argonne organized some conferences, and I know that I visited Argonne more than twice. But I can't remember exactly what they were about.

At the conferences there were a lot of papers presented, and people were discussing things, many of which that I thought were destined to fizzle out, particularly in the arena of somehow enhancing the software production and portability. For a while, I thought Stan Brown's model might be a way to achieve portability. It certainly looked like that to a lot of people, and if it didn't, it was ingenious. It was so ingenious that it deserved to be looked at, and I know I was at conferences where that stuff was presented. But after I tried to work with it, I realized what its fatal flaw was: it was, essentially, a model of a machine worse than any ever built. And from then on, I think I was compelled to conclude that only a prescriptive standard for arithmetic or arithmetics-- I could imagine tolerating a few, but not many. That would be the only way in which we could achieve portability.

HAIGH: So in that sense, then, you see your interest in standardizing the arithmetic as a different response to the same kinds of issues of portability that other people grappling with the--

KAHAN: Well yes, you see, they were still grappling with things like Stan Brown's model. Their model was that computers are diverse. They're getting more diverse. If they get too diverse, let's try to write our software in such a way that it will function reasonably well, despite the diversity. It's what's sometimes called the "least common denominator" approach. That is, use only those capabilities of which you are confident that they will be present in all the arithmetics that are commercially significant. I came to the conclusion that that was hopeless, that we would then be crippling our efforts so severely that we would not be able to enjoy the benefits of some of the better-quality software. You'd get something that would work no better, at its best, than the Cody and Waite functions, which were designed with that sort of thing in mind, and probably a lot worse.

HAIGH: Now, would it have been possible to have taken something equivalent to the BLAS approach, to create a floating point API where you would isolate the machine-dependent things and code high-performance subroutines for those specific operations?

KAHAN: Absolutely, of course. We could have done that. But the BLAS worked, because everybody who built a machine knew that it had to do matrix multiply reasonably fast. So the BLAS interfaced into a comparatively hospitable environment for the BLAS. You see, with matrix multiply, the arithmetic peculiarities of the CDC 6600 or a Cray or IBM machine, or whatever are relatively innocuous. Assuming that every arithmetic operation suffers a rounding error, it's every multiply and every add suffer rounding error, and, as Wilkinson had observed way back, the error that you accrue after a matrix multiply is very little affected, at worst, multiplication by a modest constant, like five. That's the worst effect of the range of different arithmetics in a matrix multiply unless you get into over/underflow problems, and then, for the most part, we felt that over/underflow was a really difficult problem. You might prefer to simply blame them on somebody's bad taste and choice in data, at least for the time being. That turned out to be a bad idea, but that was the general zeitgeist.

So for matrix multiply, sure, the BLAS were almost guaranteed to work, and the amount of help that the BLAS gave us was actually relatively modest. The BLAS, you see, were really a substitute for the fact that Fortran had no array operations that worked on complete arrays. If Fortran had had those operations, you wouldn't have needed the BLAS. But since it didn't have those operations, if you're going to provide them, you'd like to have a standard language for them. And the important thing was to have standard calling sequences so that your software could be portable. The fact that, then each manufacturer would tune the BLAS so that his machine would look best—that was a by-product, you see. The important thing was that Fortran was such a wretched language for doing programming of matrix calculations that we'd have to have the BLAS. And unfortunately, then the BLAS proliferated to the point where nobody could remember what the names were for all the functions, and in what order you should put the damned arguments. And just to add insult to injury, you had to include working space because Fortran didn't have dynamic memory allocation. PFORT tried to fix that, you see, and I regret that PFORT just didn't catch on well enough. So PFORT tried to provide dynamic memory allocation capabilities for Fortran in a reasonably portable way. I don't know why that didn't catch on. I can't remember. Maybe there was a reason. I've forgotten it. The BLAS required arguments, in addition to your inputs and outputs for scratch space. So that, and the fact that it propagated up into what became the EISPACK codes.

That was the great disadvantage, and it persists to this day. You've got as blizzard of names, and you've got a blizzard of arguments. And MATLAB makes it all so easy. Now, in Fortran, they have finally gotten around to putting infix operators between the names of arrays, so you can do a certain amount of array arithmetic. It should have happened decades ago. And then the BLAS wouldn't have been there, and a lot of other things would have been simplified. Fortran should have had dynamic memory allocation long ago. Mental rigidity is the only thing I can see that would explain the persistence of Fortran's handicaps for so long. And if that had been the case, then a lot of LINPACK and EISPACK would have looked so much simpler. Calling would have been simpler, and programming would've been simpler. The BLAS would probably not be there, because they'd be optimizing the matrix multiplies. We now call them BLAS Level 3, and so on. So a lot of things would've been simpler.

The need for the BLAS was to have a uniform nomenclature for the operations that we were going to use and develop a library and coordinate the efforts of people around the world. It started with North America, but it soon spread. Everywhere in the world we'd have little contributions, and you'd like to coordinate that. And you'd like people to use the same subprograms rather than writing their own, and that was the motivation for the BLAS. I'm trying to remember who instigated the BLAS. I think it may have been [Richard J.] Hanson and [Charles L.] Lawson. They're the guys who instigated it, and I think—I hope—I've captured correctly their motivation. Their motivation was not so much for portability as to make it possible for people to get together and share software without having to rewrite and not have different names for the same functionality, and so on.

HAIGH: BLAS provided portability performance and the ability to write structured code. I believe that was what Lawson said in his interview. He agrees with the advantages you've outlined. So in the case of the BLAS, people were prepared to add that extra layer of abstraction to get portability, neatness, and improved performance?

KAHAN: It wasn't even abstraction. It was just condensing the sizes of your codes and regularizing the names. If you have a BLAS that multiplies a matrix by a vector and adds a vector—you know, $ax + b$ would really be the way you'd like to write it—but you can't write it like that in the older versions of Fortran. You had to write out this damned loop. Everybody's rewriting the same loops. What happens if people give names to subprograms that do this, so they don't have to rewrite the same loops? If they give different names, we can't use them, so we have to standardize on the names, the calling sequences, and so on. That way, we can pool our efforts. And even if we were writing it for one computer, that would be worth doing. It's not a portability issue. It's, rather, a collaboration issue. Now, of course we also wanted to codes to be portable. We wouldn't have needed the BLAS to make them portable. We needed the BLAS to just reduce the magnitude of this collaboration.

HAIGH: And this same period of the early to mid-'70s was also when the NAG and IMSL libraries supposedly became available?

KAHAN: Yes, I think so. I don't remember the exact dates but yes, NAG in Britain, and IMSL. IMSL I think managed to get some contracts, which could be construed as government subsidies. They may have gotten contracts from the Department of Energy or the Department of Defense or someone like that to help them. NAG also got government money to help them start up.

HAIGH: So what was your opinion of the quality and usefulness of these libraries?

KAHAN: Well, initially, IMSL had better statistics, thanks, I think, to Ed Battiste. And NAG had better overall numerics, thanks, in part, to experienced people at Harwell and National Physical Lab, and so on, who had been developing some pretty devious things for quite a while. I think it's fair to say that, in many important respects, the Brits, as a whole, were in advance of Americans when it came to numerical software at the beginning—starting in the 1940s and '50s. So, for example, there were tracts from the National Physical Lab on numerical computation, and I probably have some in my office, which might be open now. There was a little blue book that came out, which was well in advance, as far as its appreciation of computing issues, of any of the stuff that I saw being written in the U.S., even though there was a lot more written in the U.S. Wilkinson's *The Algebraic Eigen Problem* was a huge book, well in advance of what was understood in the U.S. And then there's this very small book of his *Rounding Errors in Algebraic Processes*. That was an exposition, the likes of which did not exist anywhere else. It was reprinted by Prentice Hall. It was originally Her Majesty's Stationery Office, but reprinted by Prentice Hall. It had an enormous influence.

HAIGH: So you think that general level of sophistication translated into the quality of the NAG library?

KAHAN: Initially, yes. Ultimately, the balance changed somewhat. I think that IMSL also got into graphics and things like that. IMSL really is a huge concern. *Visual Numerics* is its name, now, and they have lots of other interests. And they appear to be a thriving company although, as a stockholder, I should have been privy to-- I mean, I have a handful of shares. I should have been privy to their balance sheet, but I can't recall ever seeing one. What it really amounts to is that the company's closely held by Charles Johnson and his daughter, and a couple of others.

HAIGH: Yes, I interviewed him. So would you say that both libraries were a major step forward over SSP?

KAHAN: Well, they were certainly better than SSP. Remember, SSP appears, at least to me, to have been merely a tool by which salesmen could say, "Oh yes, IBM's computers can do these computations. You see, it's listed here in the SSP." And it only had to be that good, as far as IBM was concerned, except that there were a few people who actually had a conscience, guys like Gustavson. And they felt that SSP should be upgraded, and it wasn't as if they were amply funded to do it. So they were doing it mainly because somehow they felt as craftsmen that SSP insulted their craft, ands they really had to replace it. Whereas NAG and IMSL were trying to sell their programs to people, over the threshold of resistance cause caused by the NIH factor, not invented here. To this day, there are people who will prefer their own programs, though they are desperately inferior to what you could get from NAG or IMSL or for that matter from the public domain. NAG and IMSL, of course, have been using codes from LINPACK and EISPACK and now LAPACK. And MATLAB uses codes from there, and, of course, that's the function of the packs. They're funded by government money because, God knows, nobody else but IBM and AT&T could afford it…or, now, Microsoft.

Those codes were put into the public domain, and people did not necessarily use them literally. They would often read through the codes and adapt the codes to their purposes. But at least they had not only the codes, but they also had documentation that gave some reasonable idea of why the things that were in there had to be there. That wasn't for every line; it would have been too horrible. There's something called templates, and, in many cases, people would use these programs as templates for how to write a code that does something like that. And I was much more interested in EISPACK, though I didn't have to meddle with it because Brian Smith was busy at that. And LINPACK, I think that I tried to get some of my algorithms in, and some of my algorithms were just simply too far out. Those guys just couldn't swallow it. We can talk about that if you want later. LAPACK, I definitely helped with Demmel and others to make contributions to LAPACK. I also tried to help with testing but, in each case, the important thing to remember is that as a professor my product is people. So I wasn't so interested in getting my names on the codes as I was interested in influencing the people who were working there, especially if they were students.

That has happened and has been happening. And currently, when I get in tomorrow, I'll be finding out how the current version of the Holy Grail is working and whether my exception handling stuff is going okay. But it's all going to be done by the students, and their names will be on the papers.

HAIGH: All right, so let's move then to discuss that time you spent at IBM in 1972 and '73.

KAHAN: Okay. I can't remember exactly why I was invited to IBM. In retrospect, the possibility exists that they hoped that I would stay. And several years later (I mean, like, a couple of decades later) I was having supper with one of my friends from there. I was visiting the East, and he asked why I hadn't decided to stay—since life could have been so much simpler. I wouldn't have had to teach except when and if I pleased, and I would have had ample resources and so on and so forth. My answer was because I feared I would turn into somebody like one of the guys there (whom I named). My friend said, "Oh no, you could never possibly turn into someone like him." But that was my fear.

You see, I could see these guys getting smug. The fellows at IBM Yorktown Heights had a really cushy situation at that time; I think, still. And they felt that they were the crème de la crème, and managed to do this without a great deal of self-doubt. And for certain mentalities and personality types, I could see that that was very corrosive. I wanted to be somewhere where… let's sum it all up by saying I wanted to have students who'd keep me honest. I wanted students who would challenge me. If I said something that was wrong, I wanted students who would speak up and say so, because I would rather be right than be authoritative. I'd like to get things right.

It's like fixing the oxygen sensor. I could replace the oxygen sensor, but then I wouldn't understand why the darned thing isn't working properly. What's caused the resistance? And to me, it's more important to understand something as well as to repair or devise it, than it is simply to repair or devise it. And then it works, and isn't that nice? I don't know exactly why. I really like to understand things, and in order to really understand things, you've got to figure out how to get past your mistakes. It means you have to keep a record of your mistakes, but people don't always know that you've made a mistake unless there's somebody who's going to notice what you've overlooked. And students are better at that than anyone else I know. And my students get to know that if they correct my mistakes, there is no risk whatever that they will suffer for it. That will definitely not happen. So my relationship with students is closer to what I want; that is, if a student can stand to work with me at all because I may do the same to him, you see.

HAIGH: So how would you describe your style as an advisor beyond the points you've already made?

KAHAN: Of course, to have a student who already knows what he wants to work on and has chosen a worthwhile subject, why that's marvelous. And I've had some students like that who've come to me with topics that I wouldn't have to work on myself in a thousand years, but I'll come to that. There's a list of students there, and I'll come to that. I really benefited a lot from having students work on things that I wouldn't have been working on myself. I would like to think that I helped them, too, or at least I'd like to think that I helped them more than I hindered them. Let's put it that way. But a very important reason, then, for not taking a job at IBM in Yorktown Heights, was that I needed at least a little bit of a hair shirt. And it looked as if it would be altogether too cushy there. So I don't know exactly why they wanted me to go. It may be because they wanted me to stay, maybe, and I certainly got the impression that there was a desk warm and chair waiting for me if I wanted to come back.

But we went off and occupied Benoit Mandelbrot's home in Scarsdale, and I occupied his office at Yorktown Heights. Benoit Mandelbrot, familiar enough?

HAIGH: Yes, the chaos guy.

KAHAN: That's the man; well, not so much chaos. It's fractals, which are definitely not chaotic. You mustn't confuse fractals with chaos but, in any event, yes. At one time, I think Mandelbrot did. I know at one time he thought that fractals would be a way of explaining turbulence, but it turns out that their statistics were utterly wrong for turbulence. And he found that out when he visited Berkeley, and Chorin challenged him. But still, Mandelbrot's a very clever guy, but he had some family matters to deal with in 1972 that would take him and his family to Paris. So his house and his office were going to be open, and maybe they just thought of me as somebody who would fill the slot. But certainly we improved his house. We fixed a good many things there. And although you'd have to ask Mandelbrot to be sure, I think he found his house functioning rather better when he came back than when he'd left it.

HAIGH: So I saw in the biographical fragment you'd written that you described your work there as "to establish mathematically sound and electronically feasible principles for floating-point arithmetic."

KAHAN: Well, there were three things going on. That turned out to be the major effort. I conducted a lunchtime seminar on most days, and that lunchtime seminar was attended by the two Freds—Ris and Gustavson. I think Bill Carter came to most of them, and I think his name was Walter Barishuss. I'm not a hundred percent that I've spelled his last name properly. And then there'd be other folks who floated in and out from time to time. We thought that we were planning the arithmetic for IBM's future systems. That was the name given to the project that IBM was working on. Most of the work was being done at Poughkeepsie, if I remember rightly. And we had come to the conclusion that if the market could tolerate hexadecimal instead of binary, it would be better off with ordinary decimal rather than binary, despite the fact that Fred

Ris and I felt that we were, to some extent, error analysts. And we knew that binary was really best, honest. But that's only for error analysts, you see. For humans, decimal is best.[9]

[9] HAIGH: So what was the benefit of decimal? Would it be that because people work in decimal, rounding and truncation in decimal produce the results that seem intuitive to us?

KAHAN: The big advantage of decimal has been described in my postings. It's WYSIWYG—What You See Is What You Get—if you display enough digits. You don't have to display a hell of a lot of digits because in most cases you display as many digits as you're carrying, at least in business. And in science, you display all the digits if you start to wonder what the hell is going on. But there is a point, and it comes soon, when you've displayed all the digits there are. In binary, for a number like 0.7, you think it's the number 0.7. You display enough digits, and it turns into something else, a 0.6 and a string of nines and a four, or something. Who knows? It depends on which word size in binary you're using. And that's extremely disconcerting.

It isn't disconcerting for me. I had learned long ago to read binary off the cathode ray tubes of FERUT. It didn't matter to me. And you know, when I think binary, I think entirely binary. I don't keep on converting them back to decimal. I just think in terms of binary—magnitudes and powers of two and stuff like that. But we aren't going to teach that in the schools. I mean, we could try. Teachers tried and showed that they're super-modern teachers who taught their students about binary arithmetic. Well, it's just a joke. Hardly anybody's going to use binary arithmetic. So, that's why we need decimal. For all of its disadvantages (and they are numerous), the advantage of having "what you see is what you get" cuts out at least 99 percent of the mistakes, misapprehensions, calls to the help desks, and so on. So from a cost-performance point of view, it was clear that decimal would be the right way to go. And it's not just the social cost. It's the cost all the way up and down the system. Of course, I still feel that way.

And although more than 30 years have elapsed, Mike Cowlishaw has apparently persuaded IBM to at least support the construction of some decimal hardware for some machines to be released. Now, whether that will persist or not will depend upon how welcome these things are in the market. It may be that IBM is doing this for the wrong reasons—because of marketing. If they make incorrect decisions, they sometimes make the right decision for the wrong reasons; that may be such a case now. But IBM is building decimal hardware as we sit here. Therefore, we're going to find out. Mike Cowlishaw used to be at IBM Hursley. I think he's moved to another IBM location, also in England. He's got a collaborator. That's Eric Schwartz; I think he's at Poughkeepsie. There's another collaborator. I can go and dig up the names if you really want them, but there's a very small group who are trying to implement decimal arithmetic, and we'll see whether it catches on.

A lot of people are getting onto the bandwagon because they think it's going to help them. Now, they may be mistaken, but people involved with COBOL. I'm not sure about ADA, and I'm not sure about the database community. But certainly in SHARE, there are people who think that this kind of decimal arithmetic is going to solve some of their problems. I believe it'll solve fewer problems than they think. But that is really not so important for commercial purposes because, the question is, will they buy it? And if they do, will it go fast enough? The commercial world doesn't need decimal to go all that fast. You see, in the commercial and administrative worlds, most of their computing effort goes into finding things in the first place and then putting them in right place after you've done the computation. And the computation takes a tiny amount of time by comparison. There are very rare exceptions, but this is what usually happens. And so the speed of decimal arithmetic is not critical for the acceptance in the commercial and administrative world of computing, but it's critical for its acceptance in the world of statistics and scientific and engineering computing. And that's where I want it to be.

You see, the statistics people and the scientists and the engineers, they may have had some mathematical training, but they're just as loused up by not having WYSIWYG as everybody else is. Most of them are human, so they would on the whole be better of with decimal than with binary, except in the few applications of digital signal processing. Perhaps the graphics world needs decimal like a hole in the head, but I think the general engineering packages are better off with decimal. But they won't go to it unless it's fast enough, and it won't get fast enough unless it's fast enough in software.

So it leads to some really interesting challenges right now as to how to do it so you can see a path of progress from where we are now, where decimal is done in software mostly slowly and often with ridiculous arithmetic properties.

One of the advantages of thinking these things through at IBM was that IBM was very much a vertically integrated company. So when we would talk about costs—and you have to remember, computer science is preoccupied with costs. What we could do would be to propagate the cost later of the saving of a nickel in the hardware. You see, if you truncated the arithmetic or you knocked out a guard digit or whatever, what was that going to do to software? What was it going to do to the portability of software? This is why I wanted you to read the thing for Cray; those are the kinds of considerations that, if you do things in a certain way, then there are certain kinds of algorithms that may have been developed on a machine with better arithmetic. And you'll be tempted to use them, but when you do try to use them, they'll malfunction mysteriously on rare occasions. And that type of malfunction is incredibly expensive, not to the hardware people, not even to the compiler people. It's incredibly expensive at the applications programmer level, if he's debugging. And if he misses this bug, the expense propagates up to all of his customers.

Microsoft hasn't learned that lesson, you see. Microsoft's customers have to keep on installing these goddamned service packs. Bugs that have escaped from Redmond proliferate like vermin and impose their costs on everybody who uses Microsoft's Windows and Excel and Explorer and whatever else. And Microsoft is not a vertically integrated company. IBM was, so we could propagate the costs up. We could figure out, and I would go make inquiries. I could figure out what the costs were for IBM's customers as well.

HAIGH: And other than being based on decimal arithmetic, were there other important novel features of the design that you were working on?

KAHAN: Okay. First of all, I can't say to this day a hell of a lot about it. I can't say as much as I would like to, and the reason is that, when I got to IBM in, I guess it was August of 1972. I can't remember. They said, "Oh, we can't pay you and you can't work here unless you sign this confidentiality agreement." And that agreement obliged me to keep things confidential indefinitely. And when I first came, I didn't know that I was going to be involved in the FS stuff and the decimal arithmetic stuff. It was just a break, you know? Let's get away from Berkeley for a little while, do some other things. I'll mention one or two of them later. And although I really didn't want to sign that, the alternative was that I should take my family, turn around, and go back. But we'd already rented out this house. We couldn't go back, so I had to sign the damn thing. I will never sign something like that again. I haven't. I have turned down consulting contracts from DEC and, ultimately, from Intel and others whenever they have adamantly insisted (or it's their lawyers, really, who adamantly insisted) that the consulting contract include a clause that I keep things confidential indefinitely until I'm released, until it's published or whatever. Or they find out about it from other sources.

The work that I did with the two Freds was the work that I've described, where we investigated these costs in a vertically integrated situation and made recommendations for decimal arithmetic. Those parts of the recommendations which looked as if they were just math and, therefore,

---

How can we get from where we are now to where I think we have to be, where decimal arithmetic is as commonplace in hardware as floating-point binary is? And it runs as fast, and it is about as accurate and has about the same range, maybe a bit wider. And almost everybody is using it, unless they're going to put it into embedded systems. That's where I think we have to go.

The main reason for going there is because you asked, "What is the advantage of decimal?" It's WYSIWYG, and that means there are fewer mistakes. So we can enhance the reliability of numerical computations, not just in the business administrative world, but in a lot of the science and engineering world, too.

couldn't possibly be commercially interesting, were summarized by Fred in a report that appeared in one of the SIGNUM bulletins. But the other really interesting stuff about why you do it and how the costs propagate and so on, that stuff never got out. I think a lot of it, not all of it, was written up in a report because I helped to write the report while I was there. And I looked at a penultimate draft before we left to come back to Berkeley. That was in the days when you still had secretaries who would type things, and they come off on a Selectric typewriter and read it.

Then back in Berkeley, I got a phone call from Shmuel Winograd who was the head of the math department. That was principally where I sat, even though that's not principally where I worked. And he said the two Freds had been put up for an award for this report, and he hoped that I could tell him what each one's contribution was. And I said, "Sure. Send me a copy because I haven't gotten a copy yet. Send me a copy, and I'll annotate it and tell you what I remember was contributed by the different folks." And I was thinking of the two Freds the way I think of students. And for that matter, even Bill Carter, who was a good deal older than I and more lubricious and so on, and it's just my habit. So I thought, sure, I'd mark down what the students had done and then let them figure out how they want to divvy up the prize. And then I got this phone call—an apologetic phone call from Winograd. He said, "You are no longer working at or for IBM and, therefore, we cannot send this document to you. It's been classified." And he went on, "I have asked when it will be declassified, as are most mathematical tracts. Most of the mathematical stuff they work on gets published in the usual way. And," he said, "I've been told that thee are two circumstances under which something can be declassified. One, IBM accepts the recommendations and declassifies the document to publish it as a justification for what they're doing. Second possibility, IBM rejects the recommendations, has no intention of doing anything like it, and doesn't care who knows. Then you can publish it for all IBM cares. And then," said Winograd, ominously, "I have been told that neither of these things will ever happen." How do you like that?

I can't reproduce now the way I felt about that. It meant that the work was going to be suppressed. I can tell you only that it made an impression upon me that was deep enough that I will never, ever put myself or the people I work with in a situation like that, never. I'll never do that again. It was a very serious mistake I just couldn't appreciate properly at the time. The stuff I do is destined for the public domain. I'm paid by the people of the state of California, and I can keep things confidential for a while. But I'm a professor, and it is not my job to keep secrets—quite the opposite.

Well, okay, that wasn't the only thing I did. It was the most important thing. It was what consumed most of my time and the time of the guys who were working with me at these lunchtime seminars. I spent time scuttling about asking people various questions and trying to elicit the data, the information, about how the costs would propagate, if they knew. Of course, most didn't know, so we had to make estimates.

There were two other things that I was doing there. One had to do with Scratchpad. Now, this was Dick Jenks' baby, for the most part, and he had one or two other collaborators there. Dick Jenks was not one of the world's great mathematicians by any means, but he had a vision. And that vision was to assist in automating symbolic analysis—what passes for mathematical analysis in algebra. And he had a system which was called ScratchPad, which I used and helped to expose to many of its unfortunate deficiencies. The most unfortunate deficiency was something we could do hardly anything about, and that was that the input was on an IBM Selectric typewriter.

And the output was on an IBM Selectric typewriter, and you couldn't have graphics. It was later when Mathematica came out with lavish graphics, because there were EGA displays and things like that available. Well, lavish graphics had completely changed the picture, partly because you'd get nice pictures, but most of all because then you could typeset the mathematics and what you saw was what you got. Instead of having just these text strings of characters and having linearized everything so you couldn't read the damned expressions, now you could almost read them. They could begin to look the way you like to draw mathematics by hand yourself. But that was going tot be in the future, and it was a future that had to come; I figured that until that particular future arrived, it was very much worthwhile investing more effort and, most of all, more talent in that area. I argued that way with Winograd, and I think I was an articulate supporter of that enterprise—not that I was going to write such programs myself. But I would be an enthusiastic consumer. I could think of lots of applications.

The application I pursued there was to actually perform a kind of error analysis, an error analysis which, as it happens, was also pursued by Don Knuth in a slightly different way. It had to do with compensated summation, but that was just to show that, despite the terrible ineptitude of the system at dealing with inequalities, there were ways that it could be used to support error analyses. Once you figured out what the induction step was, you could get the machine to confirm that the induction step worked, you see, and so on.

The third thing that I did was to work on a problem involving generalized inverses. Now, you know that not all matrices have inverses and, in particular, rectangular ones just don't have inverses in the usual sense. But they have generalized inverses, and what that means is that you can find something that behaves like an inverse on one side, if not both. I won't go into the details of exactly how it behaves like an inverse other than to say that there are a lot of generalized inverses in the literature, and the so-called Moore-Penrose pseudo-inverse is the most prominent one. It has the most prominence, but it's not the only one. And it's not always the right one to use. And it depends very much upon a norm. If you have a way of measuring error or uncertainty, then that norm should figure in the way in which you define a generalized end verse. What makes a norm apt, if any norm can be apt for your purposes, is that all perturbations of about the same norm should be about equally consequential or about equally inconsequential. And if you can choose a norm that has that property (and there are many different norms; they all have a certain association with convex bodies), then you can bring to bear the equipment of twentieth-century analysis to help you figure out what to do. If you can't find such a norm, your situation may be perilous because then you may be unable to describe, in any familiar mathematical terms, what is important about the uncertainties or errors that you would like to attenuate or assess. And we do have such situations.

HAIGH: And were you working on a way of finding such a norm?

KAHAN: No. I was working on the assumption that, having found such a norm and it's not one of the usual norms, you would now like to deal with generalized inverse. And I was able to prove that, whenever you compute a generalized inverse, no matter which generalized inverse you choose, because they're not determined uniquely in general, it would have to have a norm at least as big as the reciprocal of the norm of the smallest perturbation that would drop the rank of the matrix whose generalized inverse you sought. Now, that's a long sentence. But that's, I think, a correct statement. That meant that if you were given data which was very nearly redundant—if a few rounding error alterations in your data were enough to introduce more redundancy than already existed—then if you attempted to solve equations with that matrix by using a generalized inverse, you'd find that the generalized inverse was gargantuan. It would amplify round-off intolerably. And therefore, what you should do is go back to your data and see whether you can, in some sense, find equally acceptable data—possibly by diddling with n figures—that would be as much redundant as possible. Reduce the rank as much as possible. This was by analogy with work that Golub and I had published about singular values. These accept that if you use an unfortunate norm, and most of them are unfortunate, you don't have singular values. So you can't think that way, but I was trying to generalize it.

So I found—it's easy to prove, and lots of people have proved—that if your data are very nearly more redundant than you think, then your generalized inverse is going to be gargantuan. And the only way to beat that is to further compress the data into a region in the space of all data where the rank is minimum for data that you regard as sufficiently close to yours. If you think of your data as a point in the space of all data comparable with yours, in that space there are sheets on which redundancy occurs. Those sheets intersect themselves, and at the intersections, the redundancy gets worse. And those intersections are themselves sheets, and when they intersect, the redundancy gets worse again. What you want to do is say, "Well, my point of data is sitting out here. Where is the most intense intersection?" And if I find out that that's so close to my data that I really can't say the difference is significant, then that's where I should put my data and then solve the problem. Then the generalized inverse will be enormously smaller, and the effective round-off will be enormously damped and so on. This is part of the theme of the document in here that says, "Conserving Confluence Curbs Ill-Condition." It's dated August 4, 1972, which was just before I went off to IBM.

HAIGH: That's a technical report from--?

KAHAN: That's the technical report number six. Well, then the question, you see, was "I know that the generalized inverse has to be at least this big. How much bigger might it be?" And I was able to prove that it doesn't have to be bigger than this lower bound by a factor worse than the square root of the rank, which, for most problems, is not terribly big. So that meant that I'd nailed it. There is a generalized inverse out there somewhere. I can't tell you exactly how to compute it, but any number of approximations will do because it's within a certain range.

That's good enough. And I was in Nirvana. I'd been working on this for years. In fact, I'd started on it in Cambridge without being able to make much headway. So there I was, feeling nicely relaxed, and I went to the library to read just for entertainment. And what would I find but a paper by Yehoash Gordon, apparently an Israeli who, I think at Louisiana State, had produced a thesis and published it on what is called the Banach space projections constant problem. Now, indirectly, that was the problem that I had just solved. The Banach space projection constant problem had been an open problem for decades. A Banach space is a space with a norm for measuring distance. You can look at a subspace, and you can ask yourself, "Of all the possible projections onto the subspace, which is the one that has the least norm? Which is the one that amplifies small variations, least?"

And it turned out that nobody knew, until finally Yehoash Gordon wrote a thesis. He wrote a thesis, in which he demonstrated that the projections constant needn't be any bigger than the square root of the dimension of the subspace.

Well, that turns out to be just what I'd just proved, except I'd proven it in terms of a generalized inverse. But the generalized inverse is a part of the projector. How do you like that? He'd even proved it the same way, by using a theorem by Fritz John. Fritz John's theorem was a very beautiful theorem. It says that if you have a centrally symmetric convex body in a space of dimension *n*, you can circumscribe it by an ellipsoid so tightly that, if you shrink the ellipsoid by a factor square root of dimension—divide the square root of dimension, and then shrink it—the ellipsoid will be inside, instead of outside. That's Fritz John's theorem, published in 1948, and that was, essentially, the theorem that I'd had to use in order to deal with this problem of a generalized inverse. And that what's Yehoash Gordon had used, too. And he'd done it for sequence spaces. He'd done it in a very general sense, for general Banach spaces, and I had done it infinite dimensional, too, I mean, a finite dimensional subspace, but imbedded in an infinite dimensional space. And I had done it just for finite dimensional spaces, but it doesn't matter. They were cousins.

Well, it didn't matter. I was delighted. I had at last gotten the solution and if, instead, I had read his paper first and gotten the solution, I think I would have been just as delighted. That was one of the accomplishments at IBM in those years. There were some other little things. I collaborated with folks on this or on that in small ways that I no longer remember in details.

HAIGH: Did you ever publish your proof, or was it superseded by his work?

KAHAN: No, it didn't really have to be published, I felt, because once the Banach space projection constant was known, then it says that therefore the existed generalized inverse blah-blah-blah. If there's just a projector, then the projector can always be factored in terms of a generalized inverse, so it all seemed obvious. And I suppose it would still deserve being written up because there are lots of people who probably don't know about it. First of all, in the numerical analytical community, the Banach space projection constant problem probably just doesn't figure in people's minds, so maybe it's worth writing up. But I don't have the same intensity of desire to write it up that I would otherwise have had. But I must say, if I can just cite his paper, that enormously simplifies the proof because then I don't have to reproduce a proof like his. So it would be a much more readable paper. And his proof is pretty good. I mean, I'd have to say I admired that bit of work. He did a good job, and my proof was messier in many ways. So I felt he deserved the palm. I don't know where he is now, though. I've just lost track. Well, that's the way things go. I got back to Berkeley, and there were a lot of other things to do. Anything more you want to know about IBM?

HAIGH: I think that covers it. I will ask you later if there were any influences from the design work you'd done at IBM on the way you approached the co-processes and the standard, but let's talk now about--

KAHAN: There was a guy I met there that year. His name is Kulisch. He had an axe to grind and brought out somewhat tattered transparencies to demonstrate what he had in mind. I think for some two decades afterwards, he was still using the same tattered transparencies. He and one of the guys at Yorktown Heights finally wrote a book, ostensibly about computer arithmetic. Of course, they don't get to a rounding error until you get a third of the way through the book. It was a goofy idea, something called a superaccumulator. And in 1973, I didn't realize how serious these two were. I mean, it was folly to do a book on this. I didn't realize how serious they were, and they still are a bit of a drag. It leads to an interesting story, subsequently.

I wasn't the only one who thought that this was a goofy idea. I think the management at IBM got such advice as they could and were told, "Yes, this is a goofy idea." Well, okay. Therefore, when a German engineer at one of the IBM plants in Boeblingen in Germany became enamored of these ideas, and thought that he would build it into IBM's hardware. He was told from New

York, “Don’t do that.” IBM didn’t want to be committed to this goofy idea, but he went ahead and did it anyway. Then, at show-and-tell time, he comes across the Atlantic when it’s time for him to show what he’s done with one of the computers in this family, he mentions that he has also implemented Kulisch’s superaccumulator. And, of course, the folks there knew that he’d been told not to do that. “Take it out.” “Well,” he said, “I can’t take it out unless you’re willing to tolerate a delay to market.”

When you say *delay to market*, you touch a lot of raw nerves, because delay to market can kill something. You see, for every innovation, there is a window that opens, and if you get through that window, you can keep it open and even widen it. It’s like a sales rise, you see. There’s sort of a rise. But, if you do it too late, the window becomes somewhat closed. That limits your sales penetration. You don’t have enough critical mass in the market, so your product ultimately dies. So time to market is a very, very critical element of the release of a new product line. And when he said it would delay the market-- he had to take it out. “I’ve got to change the manuals,” and this and that and so on. They said, “Oh, hell.” They were blackmailed. They had to keep it in. Once they put it in, now they’ve got to support it in software. The software package was called Acrith—and you know there happens to be a paper here on ACRITH: *Anomalies in the IBM ACRITH Package.* What is the date of that? Yes, 1985. ["Anomalies in the IBM ACRITH Package," W. Kahan and E. Leblanc, Proc. 7th IEEE Computer Arithmetic Symposium, 1985].

HAIGH: What was the machine or machine family?

KAHAN: 4361. And, of course, all the stuff that’s in the hardware could be simulated in the software and the rest of the IBM line. So a simulation package was put together, and that meant that anybody who had an IBM machine could simulate the hardware that was in the 4361. But now, you see, IBM is stuck with it. They have to support it into the indefinite future. We tried to explain why it was goofy to stop them from doing it, and then they were doing it anyway.

But there was one good guy, Siegfried Rump. He was one of Kulisch’s students, and he was a good guy. And he contributed good stuff, and now he’s at a technical university in Hamburg and does good things. I’m glad he got away from Kulisch. He finally figured out why it was goofy. But while he was a student of Kulisch, he was making significant contributions, and some of those contributions and contributions from others got into this package called ACRITH, which then IBM tried to sell. And that’s what that paper’s about, you see. Well, I think our criticisms stunned them again that they came out with version two, in which they cured some of the difficulties.

[Tape 8, Side B]

KAHAN: You see, what everybody would like to do is to know that their computed result is not so wrong as to be intolerably wrong. Now, floating-point computation is usually intrinsically approximate, and in order to deal with this problem, there are two recurring themes. One is to use something called interval arithmetic, which will, at first sight, automatically give you an error bound for the error that has accrued in your arithmetic—part of it, perhaps, because of round off and part of it because your data is uncertain. And that’s all done mechanically, but it has a horrible vulnerability. It’s the vulnerability exposed by the little boy who cries wolf, which is the error bound is normally too big. In fact, it isn’t uncommon to get infinity as an error bound. Well, if the error bound is very much too big, then who’s going to believe it? Once you know that the error bound is several orders of magnitude too big, you’re just going to disregard it, unless you can redo the computation in much higher precision at a price that you can afford to pay. Although the error bound is pessimistic and says, “You’ve lost 30 digits. You’ve still got six, and you only wanted five.” Then it works. And Kulisch had this theme for combining interval arithmetic with higher precision. That’s a good idea, but implementing the higher precision without ever declaring variables to have higher precision, he would give someone the illusion that all of his variables were of the same precision as he declared. And somewhere hidden behind the curtain, there was something called a superaccumulator that would actually do the arithmetic to rather higher precision. What’s more, it would do it to rather higher precision in a somewhat indirect way—an indirect way which has been known as iterative refinement or deferred corrections or whatever. It’s an ancient idea that Kulisch rediscovered and thought was new.

And the two things then were integrated, and that’s what the ACRITH package was. But in order to make interval arithmetic work at all with this scheme, what you have to do is translate the mathematics of your problem so that the solution of your problem will look like the solution to a fixed-point problem of a contractive mapping. That means that you have an equation that you want to solve. *x* is equal to a function of *x*. You’ve converted your problem to solving a problem like this stated algebraically. *x* is equal to a function of *x*. And the function has to have the property that, if you jiggle the argument *x* in the function, it jiggles the value of the function by rather less. That’s called a contractive mapping. Well, not every problem can be converted to that form. Mind you, with a certain amount of ingenuity, you’d be surprised how many can be converted. The headache comes in finding a contractive mapping. You can always redo it algebraically. That part is trivial, but you may not get a contractive mapping. And if the mapping isn’t contractive enough, the whole scheme just doesn’t work. And in particular, if you’re trying to compute the solution of a problem near a singularity, you can expect that nothing will be contractive. So, the ACRITH package came out with a lot of subprograms, which had taken otherwise standard computations and converted them into this fixed-point system and then solved them at some considerable cost. And that’s what they wanted to market.

It can’t come as a surprise that, first of all, they found some enthusiasts, who thought this was just marvelous, enthusiasts who are still enthusiastic, mainly in Europe where people seem willing to publish impractical papers more often than I’m willing, anyway. But IBM couldn’t sell it, except to a handful of people. Ultimately they withdrew support, and the whole thing died. It was an episode that was preventable… but they got blackmailed.

And this whole story is germane because I first met Kulish in company with his partner in this folly at IBM Yorktown Heights. And his partner was the guy whom I most feared I might come to resemble if I went to work there. And although, at that supper event my friend said, “Oh no, Kahan, you’d never be like him,” I think I made the right decision. I may be as opinionated as Jim Cody says. And I may be as irritating as he says because I think he may be right. But I try to have chapter and verse to support the opinion. I try very hard before I express an opinion to have. And that particular paper over there is just an example to show you yes, we had our reasons. And, of course, there was Emil LeBlanc, who was a co-author. He was actually a student at Toronto. I think he’s still at Toronto, but he was visiting. I think he was visiting here for a little while. So it was when he was visiting that we did that work.

HAIGH: Let’s move on then to discuss your next consulting--

KAHAN: Hewlett-Packard calculators. Okay, now how did that happen? Hewlett-Packard had come out with a beautifully engineered job called the HP-35, which was the first scientific calculator with all the scientific functions instead of just the add, subtract, multiply, divide, and maybe a square root. And then they came out with the HP-45, which was an improved version. It had more functionality.

But in the meantime, Texas Instruments came out with a calculator that was a great deal cheaper, and here's how they advertised their calculator. So TI had this advertisement in the papers. It was a full-page advertisement. It said, "Type in your telephone number. Now," they said, "Take the logarithm." The logarithm turns out to be a number form ten-point-something, or nine-point-something, actually. "Now hit the exponential key. Do you get your phone number back? You do on our calculator." HP knew that it was the target of this advertisement because it did that on an HP-45, which carried ten digits. You type in your ten-digit phone number, take the log, take the exponential, and the last digit or two would change but, apparently, not on the TI calculator. HP was very worried about this, because it seemed to impugn the integrity of their beast. And they were looking around to find some consultant who would work on this, and I think it may have been Bill Worley who was working for HP at that time. I think he still does. Bill Worley had been at the University of Chicago, and he had known about the contributions that I and Kuki and these other guys had made to straightening IBM's arithmetic. So it may be, though I'm not 100 percent sure, that it was his advice that brought me to the meeting to discuss this stuff, you see.

HAIGH: So is it your impression that up to this point, which would be 1974, that the design of the arithmetic on these calculators had taken place largely without the involvement of specialists?

KAHAN: Well, without the involvement of numerical analysts like me and a few others who had this sort of experience that went back with mainframes and so on. If all you did was the typical adding machine arithmetic, add, subtract, multiply, divide, principally with integers or with numbers that have a few decimal digits to the right of the point, it was easy. Even then the calculators were aberrant, as that article shows.

HAIGH: So would the designers have been doing the same kind of thing that somebody working on a computer might have done if they found any one of these functions—just find an old textbook and implement whatever routine they find in it?

KAHAN: Well, they wouldn't find it from an old textbook. The guy who designed the HP-35, and his design carried over to the 45, had been extraordinarily ingenious. He had used an algorithm which belongs to a family sometimes called CORDIC, although that's a misnomer. And he adapted it to binary, where the algorithm first worked with decimal. CORDIC had originally been designed to make it possible for really small computers used in things of interest the Department of Defense. [Jack E. Volder, *The CORDIC Trigonometric Computing Technique, IRE Transactions on Electronic Computers*, September 1959]. It made it possible for them to compute rotations of coordinate systems, which really means to apply trig functions without actually having to compute the trig function. And he had adapted the CORDIC algorithm to the calculator. And that made it possible for an extremely simple processor, with a good deal of Read Only Memory and some tables, to do scientific calculations which previously had been though just to be in the domain of the mainframes. So it was nifty. It was a very neat job, the HP-35, for all its faults—and it had lots. It was really a very nice job, and then, of course, it went to the HP-45, which was just sort of an expanded, extended version of the HP-35. And the other guys were getting into the act. What one fool can do, another can, so TI had gotten into the act using relatively similar algorithms.

And HP was now embarrassed because it appeared that their calculator was somehow defective, and they were worried about it—I mean, *really* worried about it. They thought they had a certain reputation, and it was being undermined by this calculator. So fortunately, I asked what the problem was all about, and I said, "Can you send me samples of the calculators for me to play with before I come to the meeting?" And they did. So I had an HP-45, and I had an SR51. I don't think I still have the 45. I probably still have the SR51 somewhere.

And I discovered what was happening. It's true that the HP-45's arithmetic was somewhat grotty in spots, but it wasn't that bad. But what TI was doing was clever. You see, the 45 did its arithmetic to ten significant decimals, period. Everything was done to ten significant decimals, including the internal algorithms that computer logs and exponentials. TI was doing their arithmetic internally carrying 13 significant decimals, but they only showed you ten. So that meant that, though you type ten digits in, as soon as you did some arithmetic, you had 13 decimal digits. But you only saw ten significant decimals. Well, that could hide a lot of sins, couldn't it? The TI thing was cheaper, but that's because Hewlett-Packard can't do anything that's cheap there. Their whole culture is such that, whatever they do, it's going to be expensive. So I discovered that if you did this log exponential thing seven times, then the last digit would change. You see, their arithmetic at the 13th digit was grottier, if anything could be grottier, than the 45. And because it was worse arithmetic intrinsically, it meant that it didn't take very long for the error to creep up through those three digits. Seven times was enough. So I then was able to turn up and say, "Look: everybody who looks at that ad is being fooled. They think that the TI machine is reproducing your telephone number, but it isn't. It's your telephone number with a last digit diminished by one, followed by a certain number of nines, like two nines and a digit. Then it gets rounded up, you see, so it shows up properly in the display. They round in the display, even though they don't round the arithmetic." I said, "You do this seven times, and then you're going to get something with your digit, less one, and followed by a four-something-something because the arithmetic is so crummy. After you've done it seven times, your telephone number changes. Do you feel that that's honest? Is this an honest ad?"

Well, certainly it's got to be mysterious. Somebody who doesn't realize what's going on has to find it mysterious that after he does this seven times, that digit changes. That was a shock, and now they realized that they were in a world that was not the world they thought they were in. Whatever the hell was going on, they really weren't in control of it, but I also came with a proposal to cure the problem. I said, "You can do what they do, except for one thing: in order to be honest, round every result back to ten digits even if you carry thirteen to compute it." And I said, "If you do that, then each operation, taken by itself, will give you a rather honest answer, and you can explain this log exponential thing. That's easy because when you take the log, you've got the right log. It's correct to within just a little bit worse than half a unit in the last digit of the display. Then you can say 'Now, it's that error that propagates when you take the exponential because, if we recovered your telephone number, we'd be getting the exponential not of the number that you see before you. It would have to be the exponential of something else.''

HAIGH: So was the calculator using decimal arithmetic?

KAHAN: Oh, absolutely, yes. It was using BCD arithmetic. Well, the guys who were there weren't certain that they could see how to do this, but one of the people there, Dennis Harms actually had a math Ph.D. from, I believe it was, the University of Iowa. Dennis had a very respectable enough PhD from a respectable enough place, but it was in a topic altogether different from what he was now doing at Hewlett-Packard because he was essentially a microprogrammer. I've got to tell you, it could've been hard to get a job with a math Ph.D. at that time. And he understood, I think, instantly what I was saying. I don't know exactly how long it took him, but he went back. He rewrote the microcode in a few places, and the next thing you know they were doing what I said they should do.

The first calculator that was doing what I said they should do was the HP-27, and they sent me a prototype that I could play with to see whether things were working out the way I said. Things were working out the way I said they would. The functions really did look a lot better. In fact, Dennis Harms actually wrote a little paper, I think, that got published in *The HP Journal* somewhere. But I was puzzled because on the top row of the calculator's keyboard, they had these five keys: *n*, *i*, *pv*, *pmt*, and *fe*. I said, "And what do they do?" "Oh," they said, "you don't have to know that." It was on a need-to-know basis, you see. So I looked at these things, and I finally figured out what they were doing. These for calculating the relationship among the interest rate, the initial loan value, a regular payment, like payment on a mortgage—the balloon payment, and the number of payments all totaled. That's what it was supposed to do. It didn't take me long to figure that out. I mean, after all, I had attended an actuarial science course in which such things were discussed, even if I had read in the back of the class. Then I said, "You know, if they're going to compute the interest rate, that's a non-trivial computation. I wonder if they can get it right." And, of course, they couldn't. So there were cases where I could see that, although it's a ten-figure calculator, they've only got six figures correct of the interest rate. It didn't take me long to find that.

And I phoned my contact at Hewlett-Packard. I said, "Listen, I figured out what you're doing here, but I've got to tell you, you're losing digits. I've got this very simple example. It took me only a few moments to construct, and your interest rate is good to only six figures." And the guy said, "Well, that's okay; all we need is six figures." I said, "Look, you're going to lose more." I found an example, and I called him up, and I said, "Look, here's an example where you've lost all but four of your figures, and if you want to give me longer, I'll find some where you lose them all." But the fact that he had only four figures correct, that started to worry him. And I could hear the tone of his voice change—there were pauses. Then I said, "But you know, there is a way to do this that'll get it right. And furthermore, it will liberate you from one of the constraints that you've put on the input data I see. That's an unnecessary constraint." And I described the algorithm to him, and then I sent him something.

Yes, he microcoded it up, and he found the calculator worked faster for these interest rate calculations. The answer was just fine, right out to the last digit. And it didn't have this unnecessary restriction, and everything about this was just marvelous except for one thing: they already had a warehouse full of these calculators in cartons with their manuals, ready to hit the Christmas market. What should they do? I was invited to a meeting, and David Packard was at the head of the table of this meeting. I didn't have much to say. I didn't have to because my views were being represented by one of these guys who said, "Our consultant found a bug. The bug appears to be serious. We found a way to repair the bug; the consultant provided that. All we have to do is replace ROMs. Unfortunately, that means we have to unsolder the old ROMs and solder new ROMs in. You can't just do a plug-in, but we've got tools for doing that. What are we going to do with the thousands of boxes? Should we put them out on the market and insert a little slip into each one saying, 'Sorry, there's a little glitch, and if you send your calculator back, we'll fix it'?"

And David Packard said, “No, we’re not going to sell a defective product. It’s one thing to put them out on the market when we didn’t know there was a defect, and then later we have to chase down people. It’s something else to sell something when we know it’s defective.” And he said, “We will withdraw the calculators from the warehouse. We’ll make the necessary fix, change the ROMs, and so on. So we won’t make the Christmas market. We’ll have to go for the Easter market,” so to speak, or some other subsequent market. That shot my respect for David Packard up several notches. Now I knew he was a good guy. I knew he was technically competent. I knew that he ran a company based on what I would have to call humane principles. It wasn’t just good business. It was broadly humane. But he didn’t delay in that decision. It wasn’t as if it was an agonizing decision. It was clearly the proper decision. He made it there in my hearing, and I have to tell this story because there aren’t very many people who know it. And if you wonder why David Packard had earned the respect of so many of the people who knew him, this is another one of those stories. But in the meantime, I was in. From now on, I was participating in these calculators. That was all being done down in Cupertino. The next big task was going to be the HP-92.

HAIGH: And these calculators would be before the use of standard microprocessors, wouldn’t they?

KAHAN: Absolutely! It was a custom microprocessor; it was coming off their production lines. They were buying lots of standard parts, but the processor, as far as I know, was theirs. It had a very peculiar instruction set. Nobody else had that instruction set. It was slightly optimized for BCD arithmetic. Of course, it’s a tiny little processor and drawing juice from a battery. They were using, at that time, NMOS, and NMOS was not so efficient as CMOS. NMOS drew more power from the battery.[10]

Well, The HP-27 was the first in a line of very neat, short pocket calculators. The HP-25 came out after, despite having a lower number. That became extremely popular as a programmable

[10] I brought something here to show you what the headache was there with the batteries, because they had to use nickel-cadmium batteries in order to get enough current to run their calculators. And nickel-cadmium batteries are a damned nuisance, aside from their toxic qualities (you want to keep them out of landfills). Nickel-cadmium batteries ultimately leak. Some marketing genius figured out that they should have a special battery pack—rather than allow people to replace nickel-cadmium batteries with common-commodity little nickel-cadmium rechargeable cells, or, for that matter, with alkaline cells. I mean, why did they have to be nickel-cadmium? Alkaline cells would be just fine. Rather than do that, they said, “No, no, we’ll have these little battery packs, and we know they don’t last very long. And people will have to buy them, and it’ll be another profit item” and stuff like that. This is one of the things that makes me disparage the marketing man’s mentality. Okay? It certainly doesn’t operate in the interest of humanity.

So, I’m going to show you what happens with these damn batteries. Here is an HP-34C. This is the calculator that first had the solve-and-integrate key, and I’ve left it here with a battery. And do you see the fuzz? The battery is leaking. Do you see that it’s some sort of special little gadget? It’s stupid because they made the case just slightly different. It wouldn’t have had to be any bigger. You could have replaced these with two AA cells. These are two AA cells, except they’re a little bit truncated. Instead of having a button that bumps out, they have a button that doesn’t bump out. It’s dumb. But, that’s what they did. That’s the marketing man’s mentality for you. And that ultimately was death to the calculators. My view is that if other people didn’t repeat it, you see, but it kills the calculator because your batteries leak. That’s why, you see, these batteries are not in the calculator, so I can show you I keep mine…I don’t want to ruin the calculator. I have another 34C. And, of course, the batteries go to hell, and you know you can’t buy these anymore? So what I have to do is go to Radio Shack and get cells to take this apart and-- Oh, it’s just awful. So, of course, the calculators die very soon because the batteries leak.

calculator, which the 27 was not. Peter Henrici wrote a book on numerical analysis on the HP-25 calculator, but there were still some problems. [Peter Henrici, *Computational Analysis with the HP-25 Pocket Calculator,* Wiley, New York, 1977].

The trouble was that the transcendental function library was not as accurate as it should have been, and the reason was they were using the same algorithm as they had on the HP-35. And remember, I mentioned it's a CORDIC algorithm. Well, CORDIC is one of the family of algorithms called pseudo-multiply, pseudo-divide algorithms. I don't have to describe what it's about here other than to say that, when I analyzed the inaccuracies that were occurring, I realized that they would lose very little in a way of performance if they organized the algorithm differently, so it would be an honest-to-God pseudo-multiply, pseudo-divide algorithm. That's CORDIC. It's just pseudo-multiply, pseudo-divide. And if you can compute certain crucial functions (which I called kernels), not very many, then you can compute all of them from those. And you can test it, and you'll get very nice results—especially because Dennis Harms had gotten it out to 13 figures. And the algorithms I had in mind would lose one of two of those. So when you rounded it back to ten figures, your error would be certainly less than one unit in the last place displayed, and maybe just a fraction of a unit. So I persuaded them to adopt these different algorithms but, in the meantime, what they really had was a different fish to fry. I mean, all right, it's nice to have transcendental functions and so on that are accurate, blah-blah-blah.

That's all very well, but their problem was that they had a financial calculator, the HP-80. It was inaccurate. That's true, but more to the point, it was very hard to use. People had to memorize various arcane steps in order to use it, and Roy Martin had come up with a way of seeing what these things do that would require a great deal less memorization. It would seem to be transparent, and he wanted to build a financial calculator that used these ideas. The trouble was that the equations that had to be solved would, in some cases, take an inordinately long time to solve. So using the numerical methods that he knew about, he was getting into situations where sometimes the program would actually go into an endless loop. What could I do to help? And I came up with these ways to solve the financial equations, particularly to solve for the interest rate. I took the opportunity to get all of the keys to work accurately so that your error would be less than a unit in the last place, every time. Also, a way to solve for the interest, which amounts to solving a polynomial equation of as high a degree as the number of payments. They had been working with a situation where they expected to handle, perhaps, up to a million payments. That was what they had in mind for the HP-27. And I found a way of doing that, but then one of the managers said, "No, you know, if we're going to use this for electronic funds transfer, the number of payments could be enormous. I mean, people might be calculating every second, a payment every second." So I had to revise the algorithm very substantially so that now it would be indifferent to the number of payments.

HAIGH: So in that sense, the number of payments is a function of the frequency with which interest is compounded?

KAHAN: Exactly, every compounding period you get another payment, you see, especially with electronic funds transfer. They have huge bandwidths, so you can make lots of these little payments. So I helped them with that, and that was the design of the HP-92. And there was something I wanted from the HP-92. It wasn't just to get the calculator to work well. You've got one of *The HP Journals* that describes the HP-92 in the article by Roy Martin. There was something else I wanted them to do. I said, "You know, the people are most vulnerable to this sort of stuff when they don't understand interest rates." They don't fully understand the terms of the loan, but there is a law passed by Congress, called “Truth in Lending.” It’s actually supposed to be active now, but it’s pretty much moribund because the terms are interpreted in such a way that the lenders can really lie—especially if you have adjustable interest rate loans, you can get terribly socked. But at that time, there were no adjustable interest rate loans. You contracted for a fixed interest rate for the period of the mortgage or the lease or whatever it was. And what I wanted was something that would enable ordinary people—especially elderly people, who are very vulnerable—to figure out what the actual interest rate was. If they knew what payments they were scheduled to make, especially with initial payments, points and things like that, they could use the calculator to calculate the actual interest rate, and see whether that matched what they had been told. What I wanted was that a copy of the manual be printed with big type and with cardboard pages so that one of these things could be mounted, if not in a bank, then at least in a library. Somebody could come up and follow the instructions, which are now big enough type that they can read. Now, mind you, this is 1975, I think. But they could follow the instructions in the library and figure out what the interest rate was. They could even print out an amortization schedule. They could print out what all the payments were going to be that would match this interest rate and see if that’s what they were doing and so on.

The marketing people nixed it, but they had killed us already before they nixed this idea. They had already killed this project. It was an extremely successful calculator. How did they manage to kill it? Well, here’s what happened. The calculator outstripped all expectations. Nice big keys, a printer, a big display, easy to use, really easy to use. Roy Martin had figured out what would make it extremely easy to use. It was accurate. It was reliable. If you ran various tests, like let me calculate: I’ve got the payments. What’s the interest rate? Okay, you got a number. Now, I said, “Well, for that interest rate, what should the payment be?” Hey! I clicked it, and it calculated the same payment as you had before. It was all very consistent. Some of that’s mentioned in Roy Martin’s article. It was just marvelous, and it was going to be priced at $450. And the executives, especially those in marketing said, “Hey, there are people who would pay $750 for this. We should price it at $750.” “No, no!” came up a voice from below. “$450 is the right price.” “What the hell do you guys know?” said the more senior marketing people. “This is going to be value price. $750 is well worth it. We’ve shown it to various bigwigs, and they’ve said, ‘Oh yes, absolutely. I would be happy to pay $750 for this.’’’ And it came out with that price.

But the discretionary limit for middle managers was typically $500. If something cost less than $500, they could sign the check themselves. If it cost more than $500, they had to go up the line for approval. And that’s what killed it. The middle-manager types recognized how valuable this would be. But at a higher level, people said, “Hey, don’t we have one of these mainframe computers for you?” So of course, middle managers didn’t have the discretion to buy it at $750. Well, needless to say, Hewlett-Packard finally did drop the price to, I think it was $495, or something. But it was too late. The bloom was off the rose. I don’t think stories like this are appreciated well enough in our industry, and that’s why I’ve gone to the bother of telling you about it.

HAIGH: Was that Hewlett-Packard’s first specialized financial calculator?

KAHAN: No, HP-80 was their first one, and it was very popular. It’s just it was inaccurate, and it was devilishly hard to use. But it was copied by lots of guys. TI copied the functionality and, as it happens, they copied the inaccuracy also. They copied the difficulties of using it also. It was hard to use the TI calculators, and they even expanded on the wrong answers. The TI business analyst used to go catatonic for all kinds of problems. I’d get rich at answers for others. So you've got to understand that the HP-92 was a gem in that it worked so flawlessly on everything. And you'd get a good printed record. Yes, well, so we were defeated by the marketing guys on two fronts. We didn't get them into the libraries because they weren't willing to make the big print manuals. And we didn't get them out there in large numbers when they would have been the newest and hottest thing. We couldn't get them out there because the price was too high, and lowering the price afterwards doesn't really change people's opinion. What they remember about things is first impressions, you see. You only get one chance to make a first impression. So I thought that this would be a cautionary tale.

Well, we could go on now with more of the Hewlett-Packard stuff because I had another ambition: the HP-15C. That was my ambition. I wanted to have a calculator that could be used by practically all engineering and most science students. And I wanted that calculator to be able to do all the math they were going to use up to their sophomore year except for divs, grads, and curls because on a calculator, you can't really display regions and things like that. But you could certainly do everything else. You could do an awful lot of circuit calculation. You'd be able to do an awful lot of the calculation for the distortion of elastic structures in civil and mechanical engineering. You'd be able to solve some of the differential equations in kinematics for simple cases. You'd be able to do complex variables cases, so you could plot various flows. You know, I had a fair idea of the things that they have to learn, you see, up into their sophomore year. And I said, "I wanted a calculator that they could do all that stuff." That would mean that professors like me could give them problems that were simultaneously more realistic and less tedious. And at that time, there weren't personal computers in large numbers. Personal computers were a hobby.

HAIGH: So what year was this?

KAHAN: Well, the year I formulated this plan was 1978, and the year it got realized was 1982. In between those, what I had to do was to establish my credentials, establish, really, my credibility. So they would believe me when I said, "This is the right way to go." And establishing my credibility had begun with the financial calculator. This was followed by some other financial calculators: an HP-38, where we modified some of the algorithms, and then ultimately the HP-12C, which I've got here on the table. If you would like to have one, just go down to CompUSA and pay $70, and you get one.

Actually, in one of these documents, I think I've included a price list. So you can see for yourself what the prices were when these things came out and how rapidly the prices plummeted. Here's a 1984 price list. Now, this thing here is all packed together, and my hope is that you will send it back, you see. Okay, let's look here. The HP-12C financial calculator, you see the list price, initially $120? And these guys were going to sell it to you for $90. But it only cost HP something less than $15 to make. We'll get to that. They no longer list the 34Cs because the only calculators they list here are calculators which require either the little button cells that you can replace easily or, in the case of the 71B, it runs on AAA cells that you can replace easily. The calculators that we built, which required these special packs, these special batteries, that's gone. Thank God.

Anyway, the 38C was in that unfortunate format that requires these special battery packs, and I was arguing with the manager of this group, Stan Mintz, and saying, "Listen, what you want to do on your programmable calculators is put in a *solve* key." Now, I think that story has been told somewhere. I don't know where.

HAIGH: Yes, one of the articles in the *HP Engineering Journal*.

KAHAN: No, that wouldn't have been in *The HP Engineering Journal*. It might have been an article somewhere else. I don't think HP would publish what I'd said, because Stan Mintz was a buccaneer. He had forwarded this suggestion to the marketing people and characteristically, alas, the marketing people had come back and said, "We've asked if anyone wants a *solve* key, and no one has ever heard of such a thing. They don't say they want it." And in consequence, Minz was not going to go out on a limb and build it. Now, that was a mistake. I mean, Hewlett-Packard works best when the engineers come up with ideas that have value in the eyes of the engineers, and then the marketing people have it as their job to sell it rather than have it the other way around.

But by this time, the calculator had started to become a commodity, and Hewlett-Packard had moved the calculator operation from Cupertino, where it was part of the advanced products division. It was a sort of a research-like division. They moved it up to Corvallis, Oregon, where they had both a fab line and the calculator division, which was supposed to be satisfying a sort of customer market. And the people who ran it were oriented around sales, rather than around research and engineering. So their attitudes toward these things were quite different. They were developing a product called the HP-85, which was a desktop machine that you would program in Basic. And I have one in my office still in the math department. At approximately this time, there were several projects going at HP, some of them having moved from Cupertino to Corvallis. The HP-85 was going to be a desktop machine with a nice keyboard and a display about so big, a cathode ray display about so big.

HAIGH: So you're talking about the 85 at the moment, and the *solve* key, is that relevant to what became the 15C you mentioned earlier? Or is that a different machine?

KAHAN: It got into the 34C first, and then the 15C; it got into the 34C in a way that is going to make an interesting story. I came up to Corvallis one day while we were working on some of these calculators and found the group with whom I worked—Stan Minz's group—deeply depressed because they had been told that they were losing money. And this had to be paradoxical, because their financial and other calculators were the only ones that were selling—the financial calculators were selling in numbers. And none of the other groups were selling anything much. They were all behind schedule, you see. So how could it be that this division was losing money when it was the only one selling anything? Well, that was very depressing.

Next time I came up, there was a picnic scheduled on the grounds. Finally, the accounting people had figured out what was going on. You see, they charged all the overhead expenses of this establishment—the fab line producing semiconductor chips, not just for the Corvallis group but for other parts of HP—and everything else was charge pro rata to sales. Okay? These guys were the only ones selling, so they were the only ones who were bearing all the overhead cost for everything. They were paying for the lights and everything else for everybody, losing money. They were the only ones who had an income that could pay for some of this stuff. So suddenly they turned into heroes, and Stan had scheduled a picnic for Friday afternoon on the grounds to celebrate. He had spiked the punch, which was very definitely against Hewlett-Packard policy. But I can mention this because I don't think he works for Hewlett-Packard anymore, hasn't for a long time, so I can mention this. And he was in an extremely jolly mood. As you know, I am teetotal, so I can observe this stuff with a certain equanimity. "Professor Kahan," he said, "you've been nagging us about the damn *solve* key for the longest time. I tell you what, when I went to college, I had an enormous amount of trouble with integration. Now, if you can tell us how to get an *integrate* key onto a calculator," said he, figuring it was impossible, "if you can figure out how to do that, then I'll let you have the *solve* key, too." And so a couple of weeks later, in collaboration a bit with Dennis, we came up with what I thought was a really good integrate algorithm.

HAIGH: And actually, it's the *integrate* key that you describe in a paper in the *HP Journal.* [W. Kahan, "Handheld Calculator Evaluates Integrals," *Hewlett Packard Journal,* August 1980, 23-32].

KAHAN: Yes. I describe the *solve* key, and I describe the *integrate* key. But I certainly don't describe the way they came into the systems, you see. That we will talk about when we talk about Robert Barkin. Anyway, that was a promise he now lived to regret, because he had no support from the marketing people that any such enterprise would pay, and he didn't want to go out on a limb. However, he had something else. There was a young woman he'd been compelled to employ because of Hewlett-Packard's affirmative action and nondiscrimination policies. He didn't want her, and he was stuck with her. But suddenly he saw an out. He now knew what she could do. She could work with me on the design of a calculator with *solve* and *integrate* key which, as it happens, he hadn't the slightest intention of producing.

HAIGH: And was the *solve* key one of the things you saw as a crucial feature to realize this personal ambition of a calculator that would support everything that the students needed?

KAHAN: Absolutely. Well, that had to be in there because that deals with what are called implicit functions, you see. You've got a function. The function is defined as a solution of an equation. So absolutely, and the *integrate* key was needed too. But you see, I told you when these ideas came into my mind. This was 1978. Initially, I had in mind that the *solve* key was needed for the business world. That's who I really intended it for. It was only when he prodded me into producing an *integrate* key, too, that I said, "Well, with *solve* and *integrate*—this could be a calculator for engineers."

So it's 1978 or thereabouts, and I worked with this young woman. I regret that I have forgotten her name. Her name probably figures somewhere in these documents about the contributors to these calculators and maybe if I read through them, I'd find her name and say, "Yes, that's the one." The way we worked was that each of us had an aluminum box connected by an umbilical cable to one of these calculators, like the one I've shown you in my hand, the 34C, except originally that case design was intended for other calculators first. And there were little paper labels over the keys, and we could re-label the keys. And inside were the actual guts of the calculator except that the ROM was replaceable. What she would do would be to microcode up a calculator, put the microcode in the ROM, see that it worked on her box, and send another copy of the microcode to me by UPS. I put the microcode in my aluminum box, and I'd see what she had. I'd look it over. I'd say, "You know, we really ought to do this-or-that instead." So I'd send back a letter, and also the old ROM; these ROMs were erasable with ultraviolet light and somewhat expensive. So there was this exchange back and forth, and since nothing else seemed to be happening, there's no deadline or anything, we also improved the programmability. We decided to make it easier to program and so on.

But still, nothing was happening. I didn't know at first why Mintz said, "Okay, we're going to build this now." I was told by the guys up in this group that what had happened was that Mintz had seen his engineers clustering around her desk, and had wondered about that. First, she wasn't pretty, and second, she was married. What the hell are they doing there? He discovered what they were doing was using that box. For example, if they had to design a transistor circuit and figure out what the bias should be, and there are some resistors that then have to be done to get the right bias, you've got to solve some equations. They're only slightly transcendental. There is no closed form. Of course, what you can do is you write a Fortran program. You go to the HP3000. You compile it. You get your result and so on. Everything's fine. But, they could just go to this box and key the thing in, and they got the solution almost instantly. It took much less time than writing a Fortran program. And what's more, writing a Fortran program, there is, as I explained to you, a significant probability that you're going to have to run it more than once. I mean, when I got back to Toronto from Stanford, I made some changes, which got the run rate down to four. You'd run four times in order to get one job. And I don't think the HP 3000 could have been any better than that, although HP had some good computer guys.

He also saw the guys who were designing the power supply, and it has a little transformer in it. It uses saturable core in order to get the oscillations to come out. It's got hysteresis loop. You wonder, "What is the efficiency?" Well, it's a certain amount of power dissipated, because every time you go around the hysteresis loop, you lose a little bit of energy. So they have to integrate that in order to figure out how much energy they've lost, you see. It's a simple little integral. It's not all that difficult except that it can't be done in closed form. You've got to do it numerically. Instead of writing a program in Fortran to go to the HP 3000, they'd key in this little thing, click-click-click. They wait for a little while, and out comes an answer. Not only that, the answer comes out with an uncertainty. You're told, "The integral is equal to this, plus or minus that." And where does the plus or minus come from? Well, it depends on how well you know the thing you're integrating. And they had some idea of how well they knew what they were integrating, so they had not only the power dissipation but, also, how uncertain you could be about it—which was really rather more than you could get from the Hewlett-Packard 3000.

The penny dropped in Stan Mintz's mind. That's how the HP-34C was born. They agreed to do it, and then like a thunderclap, they were appalled when I said, "You know, we're going to have to put some guidance into the manual because people who use these keys, especially the *integrate* key, they can fool themselves. These things cannot be foolproof. There will be situations where people will get misleading answers, and they need a little bit of guidance about that." "Kahan, you just told us to do this stuff, and now you tell us that you're going to get wrong answers! I mean, all this time, we've been listening to you tell us how to get the right answer, invariably, every time!" Well, the difficulty was then that I'd have to go to the manual writers. But there was a Hewlett-Packard policy which said, "We are professionals, and we sell to professionals. We tell them what the device does, and they figure out how to use it. We're not writing tutorial material in our manuals." And I tried to explain, "Look—this time you've got to put some tutorial material in the manuals. You really must. Otherwise, folks are going to fool themselves."

Well, the managers wouldn't do it, but I had persuaded Barkin, and I can't remember the name of the other guy. It's probably in there somewhere. There were two guys who were writing the manuals, and I persuaded them. I think persuasion is the wrong word. This was a case of subversion. I subverted them and got them to do something that their managers had told them not to do. The manual writers listened to my arguments and decided that I was right, and their managers were wrong. And that's a dangerous decision, you know. You can get fired for that. They wrote the two extra chapters into the manual, which said something about the *solve* key and something about the *integrate* key, and a little bit to warn you. And I had written up some more stuff, which ultimately got into the *Hewlett-Packard Journal*. And the managers were outraged.

They had said explicitly, “Don’t do that.” And now their guys had done it. And they said, “Take it out.” And they were told, remember, “If we take it out, it’ll delay the appearance to market.” You’ve heard that story before. That was it. The managers had just been blackmailed by time-to-market, so they left it in.

Then afterwards, they did a survey, and they discovered that the customers loved this stuff. In fact, the customers would often say they had bought the calculator because they’d been told that there was advice in the manual about these problems, which was advice they actually needed not only for the calculator but also when they solved similar problems on the big machines. And so when I came up with these articles, they were perfectly happy to print them in the *Hewlett-Packard Journal*. I was told by the editors some years later that they had had more requests for reprints of these articles than for all their others put together.

[Tape 9, Side A]

Session 6 begins on the afternoon of the 7th of August, 2005.

KAHAN: Okay, so they put this, you might call it tutorial material, into the manual. I believe some of the manuals are now available online, and so you can look at it yourself. Well, of course, the particular writing staff manager was pissed, and the marketing people were, of course, annoyed about a violation of policy; when they interviewed customers, though, they had these little questionnaires—you buy something; they’ve got a questionnaire, you send it back. They would see that people actually liked that material and, in some cases, had bought the calculator because the material was there, so they felt confident as purchasers that they could use it. I think ultimately there was some award from the Willamette Valley Chapter of the Technical Writers’ Association that went to Barkin and company for writing what was considered an exceptionally good manual. That was the first of two occasions when I heard about an award for an HP manual. So there were guys who looked at me sidelong at HP, but there wasn’t any hostility. They figured I had done the right thing. The calculator was extremely successful, as such calculators go.

Of course, they have a very brief lifetime because of this damned battery problem, so Hewlett-Packard switched to a different technology called CMOS. It draws very much mess power. They were using liquid-crystal displays instead of LEDs. Much, much less power, and so now they could run their machines off these little, pill-sized batteries. So what’s the next one on the list there?

HAIGH: In terms of calculators, the 15C, I think. You’ve got that next, then 34C.

KAHAN: Okay. Well chronologically, what came next was actually the 12C. After the 34C, there was a 38C, which had a very brief lifetime for the same reason, and the 38C used pretty much the same algorithms as an HP 92, but the 12C was going to be a programmable financial calculator running off these little silver cells the size of pills with a liquid-crystal display in a shirt-pocketable format, which was also extremely rugged. It was designed so you could drop it three feet from a desktop to a hardwood floor and it wouldn’t break. It was marvelous, and I’ve got one here. You can’t have it, okay? I mean, the only way you can get it is to remove it, as Charlton Heston would say, “from my dead fingers.” The 12C was recoded. We had a slower processor. Because it was running off these batteries at very low power, the processor was slower, and so we had to rethink some of the algorithms. Not all of them, just some of the iterative ones, the ones that took a while to compute the interest rate and internal rate of return, and so I did rethink the algorithms. There’s a discussion of that in my notes on equation solving.

Rich Carone did the microprogramming; his name is on there, and the 12C came out-- I don't think anybody could have predicted how popular the thing became. It became part of a standard uniform for real estate people and for people running leases and things like that. It was just so convenient.

First of all, it was extremely important for women who were in these things to have something very small that would not bulk up in their purse. It was important for men who could slip it in their breast pocket and it wouldn't bulk up in their jackets. So that mechanical aspect of it already made it attractive. Secondly, it didn't have these damned battery problems. The batteries lasted for a very long time. I've got some that have lasted for 15 years, and it was easy to program, as such things go, and it was very accurate and reliable. The telephone answering staff at Hewlett-Packard had been received calls about our financial calculators, and when the 12C came out, they were getting lots of calls. People would say, "Look, we've been using these tables for years, and they're discrepant. They differ from your calculator. So your calculator's broken." And of course, the girls would be as polite as they could and they would take this down and they'd say, "I'd have to get back to you," and they would report on the data and so on and would come back to somebody there who'd run it off on a big mainframe computer, if that's what it took, and in every case without exception the calculator was vindicated, and the girl would have to call back and say, "I'm sorry, your table is mistaken. The error is in your table." After a while, they accumulated enough of these where the girls would not have to say, "I'll get back to you." They'd say, "Oh, yes. That's a known error in your table." That was it. After a while, they started saying that even when they hadn't checked on the table. So this thing became very much a *de facto* standard throughout the industry, and it continues to be sold.

Now, I think that they've modified the 12C a little bit in some ways; I'm not sure what they are. And so it's possible that if you bought a 12C today—or maybe they're calling it a 12C+, I don't know—it may be a little bit different. But certainly up until a couple of years ago, when I last looked, the 12C in CompUSA running for $70 bucks was the 12C that looked just like the 12C that I'd done, except that it was now produced in some other place. Maybe it was Brazil, or it used to be Malaysia for a while. Other than that, it looked as if it had been persisting for some 20-odd years, which is phenomenal for an appliance. On the strength of that and a few other things, like the HP 85, it looked as if they were going to contemplate building the engineering student-oriented calculator.

On the HP 85, I think I subverted one of the guys working there. His boss, I think, who was Swiss, went back to Switzerland for a vacation, and we took that opportunity to insert a number of more refined matrix-manipulating programs than his manager had wanted. He wanted to have matrices in there, but I put in such things as zero dimensions, so that you could have a row or a column of a matrix with, let's say it's a row with ten columns but zero rows. This does sound perverse but, it turns out when you're programming, you sometimes want to start with a null matrix—null in the sense that there isn't anything in it—and then you're going to start to put one row after another into it. To a computer scientist, it would be perfectly obvious that you have to have such a thing, but to this guy's manager, it was not so obvious. That matrix package was also very successful on the HP 85. Naturally, he also had the very accurate transcendental functions and so on using adaptations of the algorithms from the other calculators. And so, on the whole, this influence was spreading through the company. The HP 41C, with which I had almost nothing to do, used some pretty ugly programming—but they used our transcendental functions and arithmetic. The ugly programming was there because of an idiosyncrasy in the guy who ran the project.

Well, when we then had these things going, it was time to build the HP 15C, and I had to persuade them to do it. Here, again, is an unfortunate story. I had looked through the catalogues for various engineering colleges in the U.S. I had tried to find all of them. In the library there was a place you could go, because students who were going to apply to various colleges could go and find the catalogues. I looked at every one that had an engineering school and copied down its enrollment; I knew from experience at Berkeley that when I used the HP 34C and students looked over my shoulder, I knew that one in three would go out immediately and use his own money to buy one of these calculators from the university store. He didn't get much of a discount, either. And so figuring that then the number of 15Cs sold would be approximately one third of the first-year enrollment; I came to some conclusions about how many should be produced, and that was part of my argument for producing this calculator. The people with whom I worked most directly also wanted to produce this calculator. The marketing people, however, had done their own survey; they came out with some number which was half of mine, and I have no idea how they got it. It could be that they just took my number and said, "Oh, he's an enthusiast. Let's just simply half it." Maybe they did that, or maybe they did their own research. I don't know what they did. This was to have a devastating impact later because, after we got the calculator microcoded and everything was done and we had it tested and running, the marketing people then told one of our guys how many they wanted produced. It was this half of the number I wanted.

Now, fortunately, the guy they told it to was Dennis Harms, and what Harms did was to build a roboticized production line in a room that was about the floor plan of this house, maybe a bit smaller. He had a roboticized assembly line which, on the end, cost Hewlett-Packard 20 seconds of human time per calculator, from unloading the stuff they purchased from vendors at the loading dock to loading the cartons full of crated computers to be shipped out to their sales outlets. Twenty seconds of human time per calculator. This included jigs that tested partially assembled components. Now, if my memory serves me rightly, this is one of the tricks they used: a partially assembled subcomponent went into the jig; it got tested. If the test failed, this component got moved by the robot and put into a bin of rejects. Otherwise, it went to the next stage on the assembly line. And these things went faster than fingers. I tell you, I saw it working. They really went fast.

What happened to the bin of rejects? Well, somebody went through the bin of rejects and looked to see whether the rejected parts had a little red dot of paint on them and, if so, they went to the crusher. Otherwise, they get a little red dot of paint, and they go back into the supply bins. Of course, it was this partial assembly that went into the bin, but it was then disassembled; the individual components on the assembly got these dots, and they went into the stacks that were going to be used. So, you see, instead of having to diagnose an assembly to see which of its parts was defective, what they did was try it again later, and if it got rejected twice, then forget about it. All right, so some good parts would get rejected, and on very, very rare occasions, perhaps, something defective might get through. But they had very few returns. Texas Instruments used to have a return rate that ranged from a third to a half of their calculators being returned as defective. I don't think Hewlett-Packard's return rate for all causes, including the fact that somebody regretted the purchase, ever got as high as ten percent. So these calculators were being produced and, my guess is, it cost them something under $15 to produce, maybe, as little as 12 and you send it off short, but it gets sold in CompUSA for 70 bucks. So there's a lot of room for mark-up along the way.

The HP 12C was successful enough that they were willing to take my advice about building the 15C, but not take my advice about how many to build. They wanted a third of my figure, and Harms did half again what they wanted, and that's what they were doing. They were producing half of my number. The marketing people had a third of my number. I shouldn't have said a half. The marketing people had a third of my number, and Harms ended up with the production line producing half of my target number, and then these calculators were disappearing off the shelves as fast as they could be supplied. MIT, for example, for a couple of years was telling its freshmen that they should buy the calculator, and it had a special deal with HP that would get them involved in a somewhat lower price.

HAIGH: What year did the calculator appear?

KAHAN: I think it was 1982, give or take. We could probably look at the manuals and find out exactly when it appeared. Well, HP never produced an advertisement for the 15C in its own right in any Western language. It might have had advertisements that listed the 15C and the 16C and the 11C and so on, but never for the 15C in its own right except for one in Japanese. I saw an advertisement in Japanese. They were selling them by word of mouth as fast as they could produce them, and when my friends and I who had worked on this went to the marketing people and tried to persuade them, "Look, set up another production line, because you want to gather your flowers while you may," they said, "No, if we set up another production line, we may end up with inventory after all. You know how sales go. The sales rise to a peak and then they go down," and these guys thought that they must be hitting the peak. They weren't hitting a peak. They were hitting a ceiling. It's a different thing. So they never did set up another production line. In consequence, the market was starved. There were waiting lists, and that window closed, and so I never did get the calculators into the hands of sufficiently many students to change the ways in which professors would issue assignments, and that was a bitter disappointment. It colored my relations with this particular group at HP. I continued to work with them for a couple of years, but my heart just wasn't in it anymore.

I helped them with a very aggressive *solve* key. I revised the *solve* key algorithm, especially for the business calculators, because the trouble with the business folks was that, normally, they wouldn't do the programming themselves. They would assign a secretary to program the calculator, and in those cases where they had to do a slightly unusual calculation which invoked the *solve* key, the businessperson would not know what the domain of this function would be. He would not know what are the legitimate arguments; in consequence, in the course of searching for a solution, the search process might search outside the legitimate domain of the function—in which case, the sensible thing to do was to simply retrace the search in order to get back into the domain. We don't ask the user to tell us what's the domain of your function, because that can be just as hard to find as the root of the equation.

But that was then built into the 28C, the 18C, 19C, they were financial calculators, I think, that persist in the 45G. I don't think you can find a solve program like that anywhere else. Certainly, MATLAB's is not that aggressive. I've had students program these things up, but since the product is the student and not the program, I'd have to say that, sometimes, the software engineering has not been done so thoroughly by the student as I would like, but it's been done well enough that the student deserves a grade and gets passed. Well, all right, so I'll find another student someday, perhaps, to do this and put it back into the public domain. Well, that was a pretty good run with HP calculators. It was 10 years of consulting, and I was very proud of most of the machines that we built.

HAIGH: Did the C in these calculator names stand for CMOS?

KAHAN: The C for CMOS, yes. Yes.

HAIGH: Yeah. So they just started adding C to the model numbers?

KAHAN: That's correct. And actually, it was CMOS processors, even though they were still using, for the early ones, the LED, Light-Emitting Diode displays. The 38 and the 38C differed; the 38 was using an NMOS processor, if I remember correctly; the 38C used a CMOS processor. What that meant for practical purposes was when you turned off the calculator, you could then turn it back on later and it'd still have what you put in the memory; with the 38, you turn off the calculator and it forgot. This could get very annoying with the program in there, you see. But with the 12C, of course, there was just no question about if you turned off the calculator and turned it back on, it would still have your programs. Indeed, people working out of certain offices would have the programs preloaded into the calculator and they'd be given a little sheet telling them how to use the calculator for certain routine calculations peculiar to that office.

HAIGH: And you spoke of what you were doing as being writing microcode, so does that mean that--?

KAHAN: I didn't write the microcode. I wrote the algorithms, and I wrote a program, typically on the HP 85 or precursor—there was an HP 825, or something. I didn't use university resources, you see, so I would use calculators or computers supplied by HP. I'd write out first the algorithm, then I'd write out a typical program to show how the thing worked, and then these guys would microcode it into the calculator.

HAIGH: But I was wondering just about what *microcode* meant in this context. Was the regular instruction set the thing that the user would feed into the calculator and the microcode the instructions used to execute this internally, or was there a level of ROM that held regular instructions…?

KAHAN: Every time the user pressed a key, what he was doing was instigating the traversal of a piece of microcode that performed the function appropriate to that key press, and it was called *microcode* because it was the object code for a microprocessor—a very crude, rudimentary microprocessor—and in some cases, they would develop a higher-level language in which they could write the program in a more natural language, and it would then get translated into microcode. Very often, because ROM space was precious, they might have to redo the microcode by hand and hope that they could make it tighter and squeeze somewhat more so than a compiler would at that time. I have to say that modern compilers running on big machines nowadays have produced codes that are tighter than most people could possibly produce by hand, but that's not the case for all compilers.

HAIGH: So that's not quite the regular meaning of *microcode*, then, I guess.

KAHAN: Well, the processor itself came with a ROM that had a certain amount of microcode on it, and there was more microcode on a separate ROM, which personalized the calculator for different models. And so it looked just like the microcode that you would write if you had a processor whose rudimentary instruction set is microcoded to emulate a more sophisticated instruction set. There really wasn't a heck of a lot of differences.

HAIGH: Now, your career as a consultant to Intel was running parallel with this work on HP calculators?

KAHAN: That’s correct.

HAIGH: And I understand that began in 1976.

KAHAN: That is correct, too. Initially I was brought into that project by John Palmer because he had attended some of my classes at Stanford in 1966 and figured I was the right guy to ask about computer arithmetic in a larger sense than merely building the appropriate circuits. They had lots of guys who would do that at Intel, but the question was, “*What* arithmetic should we build?” When he called me about this and we chatted about the prospect, I said, “Why don’t you emulate IBM’s arithmetic, because there’d be a big market for that, with so many mainframes and so much software running on it.” His response was immediate, “No, we want good arithmetic.” So I said, “Well, the DEC VAX has got pretty good arithmetic, you know. You could use that.” And he said, “No, we want the best arithmetic. And what’s more,” he said, and this was the crucial thing, “we want it for a mass market.” When I asked him how big the mass market was, he couldn’t say—but I could get some idea because, you see, the marketing people at Intel didn’t think that there was a hell of a lot of market for a floating-point coprocessor. Certainly none of their surveys had indicated an intense demand for this sort of thing, and so Palmer said he’d strike up a deal with them. He said, “I will forego my salary here at Intel provided you give me a dollar for every one of these that you sell.” So the marketing people went way back in a huddle and, lacking the courage of their convictions, they decided not to take him up on that deal.

But there was a second consideration. When we started to shape up the specifications for what became the 8087, these specifications were going to be mimicked on the i432, which was being developed up in Oregon. The people of the 432 said that they could program anything, so don’t worry. They were building at that time the world’s most complicated computer—without a doubt—and the complication of a floating-point unit was really not a big thing compared to the complexities that they already had to deal with.

HAIGH: The most complicated computer, or just the most complicated microprocessor?

KAHAN: No, the most complicated computer, bar none. See, they were trying to build it to be extremely reliable so it would be used in the telephone industry, which meant that they wanted it to be hot-pluggable. So you would have a number of 432s around, and they would all be responding to various inputs and outputs and interrupts and stuff; if one of them started to go bad, what you wanted to be able to do was pull it out and plug in a new one. There weren’t very many computers of any kind whatever that had that capability, but it meant that the 432’s architecture was complicated, and they wanted to have various communication capabilities, and I can’t remember all of the stuff that they wanted to put in there. In the interest of reliability in the application that they had in mind, they put an awful lot of stuff into that processor, which slowed it down.

In consequence, they were vulnerable to an attack, mainly by one of my colleagues, David Patterson, who at that time was enamored of what he called the RISC (Reduced Instruction Set Computing) concept, and he regarded the 432 as the antithesis of RISC. He got students to run benchmark runs comparing the 432 with the 8080 or the 8086 on various tasks—common enough tasks, but not the tasks for which the design was optimized. Nonetheless, they published these things, and it cast a pall over the future of the 432. Maybe that pall was deserved, because the 432 was an extremely ambitious project, and its performance was certainly significantly less than had been anticipated and, alas, promised. So maybe the 432 deserved to die, but if it did deserve to die, I would regard Patterson’s stuff as an unkind cut—unfair. They didn’t try to run the things on the 8086 or the 8080 that the 432 was designed to do. They didn't even consider that, so I think it was an unfair test. Anyway, the guys at Intel thought it was an unfair test, too, but nobody protested and it was to no avail; the fact was the machine simply didn't succeed.

But the 8086 and 8088, they succeeded. The 8086 and 8088 architecture were designed by Stephen P. Morse. Morse had been told initially to build a better microprocessor architecture upward-compatible from the 8080. He protested that that was a bad idea, because the 8080 had been designed for an extremely tiny memory capacity and had, therefore, very limited registers. Intel's management felt that they needed something that was upward-compatible, and that seriously constricted Morse on what he designed; finally, he had to report to them that, with these restrictions, the design was a dead loss; it just wouldn't go anywhere. And so they relaxed some of the restrictions. They allowed him to do things that were not necessarily upward-compatible from the 8080 instruction set, but time constraints meant that he couldn't do the decent thing. So he ended up with a very limited addressing capability. He could address at most one megabyte, and he couldn't even address that without using what are called segment registers. It was an awfully grotty architecture. The registers were, in many cases, dedicated. The things that you could do in this register that you couldn't do in that one and so on, it was just a pain in the neck to program that darn thing, but that was his inheritance from the 8080.

And then we were going to have to produce an 8087 that would be compatible with that, and a coprocessor way of doing it was the only way of doing it. You asked whether the coprocessor concept was new. I don't think it was, although previous coprocessors had really been boards, so for example, the DEC VAX, at that time, if you wanted to do floating-point at all, you would plug in a floating-point board. It would then act as a coprocessor. It would extend the instruction set. But the phrase *coprocessor* is a bit vague, because if you sharpen the definition enough, then you can say this was a novel idea. You know, what you do is you tune the definition to suit the coprocessor.

HAIGH: So this was the first time that add-on floating point hardware was used for a single-chip microprocessor--

KAHAN: No. Intel and AMD had, I think it was called an 8032; they had another coprocessor. It wasn't a very good one, and AMD ended up as the only ones producing them. I think National Semiconductor, perhaps, had a really awful floating-point coprocessor. These were things in which you would send instructions to certain memory addresses that had been captured by the device. Instead of going to memory, they'd go to the device's op codes and you'd have operands which go in suitable places. This whole thing would have a little microprocessor that had been programmed to perform arithmetic, and some of it was fairly elaborate. In most cases they were just really glorified calculator chips, but all of it on one chip.

John Palmer had hoped that the coprocessor would have everything on one chip. It would have not only the floating-point add, subtract, multiply, divide, and the registers in which these operations were performed, but it would also have the transcendental function library, the math library, it would have the decimal boundary conversion of the whole thing so, when you plugged it in, everybody would get the same good quality results as everybody else, regardless of the programming language they fancied. Previously, the libraries were, in some senses, attached to the compilers and therefore attached to the languages; he just wanted to have, this as the fundamental arithmetic engine. Everybody uses it; they all get the same results for the same operations, regardless of the language in which they're uttered. And he had one other thought, which was, of course, that initially there weren't all that many compiled languages, and so the assembly language programmer should be able to get this stuff without having to write his own library.

But it was going to go for a mass market, and the only intimation I had of the size of the mass market was when Palmer had said the marketing people were unwilling to do it. I had in mind some several hundred thousand. I didn't have in mind some tens of millions or maybe even hundreds of millions. It's somewhere in that region, and that's what we've got in mind now, you see. Still, I had to decide for the multitudes, and I knew they weren't going to be error analysts. That exerted a very substantial influence on the way I was going to do things. On the one hand, I wanted an arithmetic that would have a certain mathematical integrity that would make it possible to prove things correct. That is the prescriptive idea, instead of Stan Brown's descriptive idea.

Proving things correct—well, I don't expect every numerical software item to have its correctness proved, but it's nice to have at least some of them proved so you don't have to worry about them. Concentrate on the truly unknown, on the deservedly uncertain, rather than waste your time on things that don't deserve to occupy your time. So I'd want some things to be proved, and in order to be able to prove things, you're going to have to have arithmetic that's reasonably regular. Of course, it was easy enough to do an arithmetic that was reasonably regular. The hard thing was to see not only whether you could actually implement it on the 8087 (which was going to be microcoded with a very simple internal engine) but also how it was going to fly on the high-performance processors. So I had to make sure that there was a growth path. This was not, strictly speaking, part of the consulting job. I became part of it when I discussed these things with Palmer and others and we agreed that in due course—and maybe a lot faster than we thought—the microprocessor was going to be the predominant engine, even in what were called *mainframes*. That could be that the mainframe is going to have a microprocessor and, maybe, quite a few of them—the channels are going to have their own microprocessors, so we can imagine that computer is just simply chock-full of microprocessors, and maybe then it would be a good idea to design for the distant future. So that went into the mix.

HAIGH: Were you doing that because you thought specifically that future processors would be upwardly compatible from the 8086, or was it just that you wanted this arithmetic to be something that Intel could put on its subsequent architectures?

KAHAN: Well, you have to remember that we didn't know that the 432 was going to fail, and so we wanted the arithmetic to be the same. In any box with the name Intel on it, you got the same arithmetic. That was the way Palmer had sold it to me, and it's right.

HAIGH: So you weren't sensing that the 8086 architecture was going to dominate?

KAHAN: Oh, no. If I had known that, it would have curdled my blood. It had a horrible architecture. No, but the arithmetic was what I was designing. Then a problem arose, in that they found that the op code space was somewhat tight; they didn't really have space for op codes that would use a regular register architecture. They didn't have enough space in the op code to have two registers routinely. It looked as if they could have in many cases only one register. They can have only one register; the other must be implicit. That means you've got to have a stack, so everything goes to the top of the stack. It's a goddamned bottleneck in performance, but it does allow you to get by with this op code limitation.

HAIGH: When you say there wasn't enough space in the op code, do you mean that there just weren't enough bits in the instruction format to squeeze in the codes for two registers?

KAHAN: Well, if you squeezed in more bits, you'd have three more bits, and that could mean that your op codes would now extend a byte or two longer, certainly at least a byte longer, and these were variable-length instructions to begin with.You see, the instruction has an op code and it has some argument designators, and is the argument in the register, or is it in memory? And if it's in memory, how are you going to index it? Because you don't just get things by specifying a physical address in memory, normally. Either you specify relative to where you are, especially if it's a constant, or else you have to specify it relative to pointers which are carried on other registers. The instruction becomes a fairly complicated thing to decode in order to take account of the different addressing modes—the different ways of constructing addresses—and their instructions were already variable-length instructions, which is a bad thing. It makes the instruction fetch and decode more complicated, certainly contrary to the RISC philosophy.

They were committed to an architecture, in which the operands could be in memory, instead of going with a simpler architecture, in which you must fetch operands from memory and put them into registers first before you can act on them. But that was forced upon them because, at that time, there wasn't a hell of a lot of space for registers on the chip. How many registers could you have? They were stretching it to put eight floating-point registers on the chip, especially because I wanted the registers to be rather wide. That was a bit inconvenient, too, so we settled on a stack architecture and, actually, the stack architecture had certain advantages in that. For assembly language programmers, the stack was easier to use than a flat register set, and that had already been proved for practical purposes in a language called Forth, which was a stack-oriented, recursive language. So when the coprocessor came out, the guys in Forth loved it—or they thought they loved it until they realized that the stack was finite. I'm going to come to that. That was one of my blunders.

So I thought, "Okay, we will be able to get assemblers and programmers onto this chip really fast; the compiler writers will like it because they don't really have this register maintenance headache: "What is in which register, and what happens when you run out of registers? You've got to spill things and then reload them and so on," and I imagined that we'll get rid of that by using a stack. But the stack, strictly speaking, is a wretched way to do things; I persuaded them to allow downward-addressing in stack, which meant that not only could you combine something lower down in the stack with the top of the stack, so you could reach down into the stack to fetch operands. But also, you could store something from the stack down into the registers, and you could do swaps. So what that really meant was that, with that capability, I could see that, although the programming would get tricky, you wouldn't really be severely limited in what you could do by the fact that you had a stack instead of a flat register set.

But it was extremely important to make sure that stack overflow and stack underflow would not be a serious problem. Stack overflow, when you push more things on the stack than eight, and so something has to pop out the bottom and go somewhere; and similarly, stack underflow, when you consume more things on top and you've got to somehow replenish the stack from somewhere. So I came up with a scheme for doing it, and I described that to John Palmer, and he seemed to understand it.

The implementation was being done by a team in Israel. It was being done there because, as our design emerged, the folks at Santa Clara figured that this was getting out of hand. It was much more complicated than anything they had done before, and they weren't all certain there would be a big market for it because, of course, the marketing people had said they haven't noticed a great deal of demand for floating-point coprocessors, even amongst the ones that they'd produced; they weren't big sellers. And so the guys at Santa Clara were disinclined to get involved. The folks in Israel, on the other hand, wanted to prove their manhood. They wanted to show that they could handle anything. I don't know if you've met Israelis. You can understand, some of them are a little bit like this. So Palmer was communicating our design decisions to the folks in Israel.

Alas, it is not possible for someone in Santa Clara to be talking to someone in Israel at an hour of the day when both of them are awake. The communications were imperfect. The guys in Israel didn't quite understand what Palmer was talking about, and Palmer may not have realized that they didn't quite understand it; they finally concluded that whatever it was that Palmer had described (I'm surprised they understood it) was impossible, and they communicated this to Palmer and said they had a much simpler scheme in mind. "Okay," says Palmer, "if they can't do it, they can't," and he reported back to me, "The folks in Israel can't do quite what I want about the stack, but they had found a simpler scheme." I was busy with the microcode for the trig functions, which was based on pseudo-multiply, pseudo-divide schemes. I was officially deep in that, that I said, "All right, maybe they've got a better idea. That's okay with me." I told Palmer, "Make sure that they write the software for handling stack over/underflow before they implement it." Palmer said, "Of course," and that's what they didn't do. And I didn't follow through. I made a terrible mistake: I trusted them.

Well, these were good guys, you must understand. It wasn't as if I was dealing with louts whose work had always to be looked over. For example, when they saw how much microcode they had, they would implement our specifications. They figured that they couldn't get that all into their transistor budget, and so they had to find a way—and they did find a way—to store two bits per transistor by sizing the transistors. It meant that they had to have some rather nifty sensing circuitry. Ultimately, the densities improved. I mean, density improves by a factor of two every 18 months, so it didn't take very long for them no longer to need this circuitry. In later production runs of the 807, they used ordinary ROM, but for the initial ones, they had to store two bits per transistor in the ROM, otherwise we could never have fit it all in.

HAIGH: Did it have a higher transistor count than the 8086 itself?

KAHAN: Well, I think it must have. I don't remember actually doing that comparison, but I can't imagine that they had more transistors than an 8086. I don't know how they would have gotten them. Even with a cache, the sizes of caches they had at that time. We had, to begin with, these eight floating-point registers, each with 80 bits and three tag bits. I was the one who urged them to have tag bits. I had more uses for the tag bits, in my mind, than the guys in Israel had, and so the tag bits on the end turned out to be mostly superfluous.

HAIGH: You know, I don't think the 8086 had a cache at all. I don't think that showed up until the 386 era.

KAHAN: No, internally, I guess-- I don't know that the 8086 had a cache, either.

HAIGH: I don't think it needed one, because I think RAM could keep up with the processor.

KAHAN: Initially, yeah.

HAIGH: It was the two--

KAHAN: Yeah, but I don't know. You know, there was the 80186 and then an 80286; I thought the 80286 had a little cache in it. I don't remember.

But in any event, if it had a cache, still, I don't see how they could possibly have had as many transistors as we have in the 8087. But I never actually made that comparison. Certainly, the 8087 ran pretty hot at first. I have that some of the early ones were dissipating well over two watts. I had to have a little cooler stuck on them. Later production, they didn't need any cooler. Mind you, I think Cyrix, when first produced their coprocessors, they had to have coolers on them, and then the later ones didn't. I remember I had a Cyrix chip with a cooler with fittings on it. Anyway, maybe it was on the Cyrix 486 clone rather than on the 387 clone. I can't remember, and I'd have to go and actually look at my box and see.

Well, okay. We made a really bad mistake in just that one area. Everything else on the 8087 worked the way it was supposed to. It was extraordinarily easy to use its arithmetic. We got unsolicited letters from all sorts of folks. I remember one from a woman who worked at NBER, the National Bureau of Economic Research, near MIT, and she had been doing programming for them for the longest time, and then got a microprocessor development system with an 8086 in it, an 8087, and sent me a gushy letter saying, "My goodness, this is so much easier," still programming in assembly language. "So much easier. So much of my youth had been wasted," she said, "on arithmetic crap, and now I don't have to think about those things." I don't think she used the word *crap*.

So the arithmetic was fine, but the stack wasn't. The stack was going to come to haunt us. I think I printed out the stuff on the stack, if you'll just let me go, I'll go fetch it off my machine and tear it off and give it to you. You won't care if I give you a blank sheet, here. But, yeah, the idea was to have a memory image of the stack spaced periodically in memory so that, at any given moment, the extension of the stack began at one of these images. But the tags were supposed to tell you which entries on the stack had already been aliased in memory. So when you reload the stack from memory, if you ever have occasion to do it, you just reload the stack from one of these images and copy the whole damn thing to fill it up, even if all you thought you needed was one item—because if something's been aliased in memory, then when you push something on the stack and push something out the bottom, if it's already in memory, it isn't the stack overflow. You don't have to worry about it.

HAIGH: Now, when you say, "in memory," do you mean in memory on the 8087?

KAHAN: No, in memory of the 8086. You've only got eight registers on the 8087, and you reference them stack-wise from the top, and your instruction set only gives you three bits, so you can only refer to eight registers. You could put 16 or 32 or whatever many stack registers on top of the 8087, but it wouldn't do you a hell of a lot of good because you can't refer to anything past the eight, you see. Now, the idea, then, is to write your program as it everything was on the stack, but you can only reference the top eight. If you want to reference down any deeper, then you've actually got to do something that's going to get you, then, to actual memory. With floating-point, that would almost never happen. It's very, very unlikely, and you could always fake it by doing a copy. If you know you're going to reference something down deep on the stack, in effect, what you do is you copy it and you push it later as an argument onto the top of the stack, and you do that from your program rather than from the automatic system.

So the automatic system would have made stack over/underflow extremely rare by comparison, even though it would frequently be pushing something out the bottom of the stack. Once that thing got into memory to begin with, then you could reload it. That won't happen very often. Now, if you push it out again, that's okay. You don't have to do a trap. I was so exasperated. The other thing about the floating-point stack was that, when you say the word *stack* to people, they think of what's called a *system stack*. The system stack is the thing on which you push the addresses—not values, the addresses—of operands, and you push the return address, and then you go and you call some program, and it finds on top of the stack everything it needs, including the return address, which it saves somewhere if it has to. Or when it has consumed all the operands on the stack and has put out its result, it then knows where to find the return address and go to that.

Now, the trouble with the systems stack relative to the floating-point stack is that the depth of the system stack fluctuates frequently and deeply. You're always pushing things on the stack and plopping them off the stack, and the level of the stack is rising and falling through quite a large range. I mean, it's like the tide in the Bay of Fundy. But the floating-point stack jiggles just a little bit, you see. It jiggles very frequently, but it jiggles just a little bit, typically. So it's a very different behavior, and it's on the basis of the difference in behavior that it's worth having a floating-point stack at all. Otherwise, we'd just be in the woodwork. So I thought my stack design would work well, and it wasn't as if I was the one who had advocated that there be a stack. Remember, that there be a stack had come from some other guy, the language guy, who said, "We're running out of op codes, but it's okay. If you do it with a stack, you should be able to do most of this stuff anyway." I had endorsed the idea of the stack, because I saw what I thought were advantages. I said, "Sure, stack instead of flat registers." You know, flat registers had been the way we were going. IBM 360 had flat registers, and the DEC VAX had 16, I think, flat registers; I don't remember the exact number. So flat registers were the way people were going, and later on, a number of registers got up to numbers like 128.

But here we couldn't do that, and I thought, nonetheless, the stack would be okay, and it would have been, if…. Well, now compiler writers had a headache, because compiler writers had to use a different algorithm for maintaining an awareness of what was on the stack—different than for flat registers. So that was their problem number one. If the stack had worked the way I wanted it, they would have just ignored it; they would have just pushed things on the stack as if it were bottomless. Because the cost of stack over/underflow was now enormous with the way the Israelis had implemented it, they had to prevent stack over/underflow for all practical purposes. When Morse tried to write a program to make the stack look infinite, he succeeded, but only at the cost of a program that was really horrible. And it was slow. This discouraged a certain amount of intelligent use of the stack.[11]

[11] The Microsoft people found a bizarre way of dealing with the stack. When they programmed Windows NT, a relatively recent version of Windows, they programmed it initially on the DEC Alpha. The DEC Alpha was, at its introduction, without a doubt the fastest microprocessor; the DEC people accomplished that speed by omitting a number of intermediary registers that had, in other architectures, been required. So when you had a result, you put it into a register before you moved the result to somewhere else, and these guys had gotten rid of the intermediate. But, of course, that meant that they had to be very careful about the timing. They had an enormous amount of hand-design in the machine, but they compelled the industry to follow them because they got the speed—so the industry had to use automated tools which took a while to develop. So the DEC Alpha was, for some time, the fastest processor around. Windows was going to run on the DEC Alpha, thought the Microsoft people.

It didn't work out that way, and it wasn't really the fault of the DEC Alpha; it was rather that the applications programs, which could have used the DEC Alpha, were not being programmed for it. There was such an enormous accumulation of software, especially in object code—software whose original authors had now disappeared. It really

What do you do if it's a stack over/underflow? You generate a runtime message that says, "Your floating-point expression is too complicated. Please break it up and recompile." Now, as a user of the program, you would be somewhat annoyed to get a message like that. But if the programmer in debugging his code has actually exercised all paths through the code (and if he's conscientious, he really should though, to be honest, in some codes it's not practical; the number of paths is astronomical but assuming that he's exercised enough of them) he would discover that, for some path in his code, he's going to be pushing too many things on the stack. Not that that's the way he thinks about it. He's going to discover that he's getting this message, which tells him he put too many things on the stack, so break up your floating-point expression. After he's done that, he's not going to get the message, right? So the user won't get the message. Everything is lovely. But what happens when the invalid operation is something like zero divided by zero? Well, you got a trap. You go to the trap handler, the trap handler says, "This is not a stack over/underflow," so I am now going to go to wherever I have to go to deal with zero over zero. Well, the guy says, "Zero/zero," oh, you know what you do with that: you raise the *invalid* flag, and then you resume computations. So you resume, but when you resume, you go back through the handler that looked if it was stack over/underflow, and what that handler has to do is now go back to re-execute the code that was trapped. But it must enable the trap in order to catch the next stack over/underflow, if one occurs. You can't enable the trap if the flag is up. If you do that, it'll just trap again; it has to put the flag down, and then you resume code. This means, in consequence that, for the applications programmer, you expect it that if he did a zero over zero, then the invalid flag could be tested later. He's not going to get it. The invalid flag will never be raised, so you can never detect these invalid operations.

[Tape 9, Side B]

KAHAN: One of my friends in programming discovered this horror, and that's how we diagnosed what was going on. This is an example of the problems that were precipitated because I didn't follow through. Well, you can blame the Israelis; you can blame John Palmer; you can blame the difference in time zones—but you can also blame me, because if I had followed through, this would not have happened. I think that that has been a significant factor in delaying

---

wasn't practical to change it. Even recompiling was unreasonable because you can't just recompile it for the DEC Alpha. The DEC Alpha had a fundamentally different architecture, different ways of managing things, so just recompiling by itself would not work, certainly, for the system code and for a lot of the code that would have to be recompiled…the drivers, for example. So the volume of available applications to run on the DEC Alpha simply didn't materialize fast enough; therefore, Windows NT had to be replicated on the old Intel architecture, and they had to transfer as much as they could, including the compiler.

Now, the trouble was that the DEC Alpha had a flat floating-point architecture, and 8087 had a stack; they didn't really want to have to redo that much of the compiler. It would have been a fundamental change in the compiler to handle the stack instead, so they came up with the following idea. Pretend the stack is infinitely deep. That's easy, because when you strip out register management rather than trying to put a new register, you just knock it out. After all, there is a certain stage at which, in many languages, you imagine that there is a stack. It's later that you have to translate the code that pretends there's an infinite stack and the code that pretends there's flat registers; wherever you put too much stuff in registers, you get some spill, as it's called. You spill things out. There's an algorithm for figuring out what should you spill out and then what should you put back. Chuck it all. But then what happens if the stack overflows? Oh, that's easy. You enable the invalid operation trap, which is what is precipitated when the stack overflows or when it underflows. The trap handler will now look to see whether the invalid operation was precipitated by a stack over/underflow. If so, it's going to handle it. Otherwise, it'll pass it on to some other handler if another handler exists; if the other handler doesn't exist, then you'll just simply do whatever you would have done at the trap, then disable it—a mash, is the word that Intel uses.

the full support of the 8087’s capabilities, because all you really need is one black spot on the flower and it doesn’t look pretty anymore. There was that black spot. Okay, next topic.

HAIGH: Oh, one of the things I saw on your description of this was that you had managed to prove the transcendental functions with 14,000 inputs, so I was wondering, was that--?

KAHAN: That was very unusual.

HAIGH: Did you have to do all kinds of clever mathematical reasoning to figure out that 14,000 inputs would be enough and exactly what they would need to be?

KAHAN: It wasn’t all that hard. Because of the way the pseudo-multiply, pseudo-divide algorithms work, there is a certain hierarchical structure. Once I could prove monotonicity—and I did; I chose my formulas to make sure they were monotonic, and there’s something a little bit about that in the document on the monotonicity of some computer functions. Once I knew that everything was monotonic, then it turned out that it followed, assuming that the add operation carried properly, carry propagations was okay, and that, therefore, you can test multiply, and then you can test divide. So they worked, and you tested the transcendental functions. The test of the transcendental functions was based on the hierarchical structure of the monotonicity of these operations which meant, roughly speaking, first you tested for very tiny operands, and once you’ve tested for very tiny operands, then operands that are bigger than that gave you things that are simple transforms. The bigger operands could transform in a simple way into tiny operands. You already knew the tiny operands are going to work, so you just wanted to make sure that the transform works. The transform was of an extremely simple kind, and if it worked at both ends of the range of the transformation, then it had to work in between, because the rest was just add/subtract. So there you built up this hierarchy that says, in effect, once you knew that add/subtract—and, therefore, multiply and divide—worked, and you knew the things were monotonic, then tested these operands in this order, and if everything’s correct for those, everything had to be correct in between.

So we did it—and it was. And that was the end of that as far as I know, except when people made a mistake in the microcode, and that had to be fixed. Once the microcode was operating faithfully, no one ever found a bug.

HAIGH: And were you involved in the development of the 80287 and 80387 chips?

KAHAN: 80287, no. 80387, yes; I worked with Jim Valerio. 80387 had somewhat better trig functions, and it may have had other tweaks in other places; trig functions were the main difference. It produced sine and cosine simultaneously; it did a few other little things. I actually went up to Hillsboro to work with Jim Valerio on this stuff. I met a few other really interesting folks. There wasn’t really a great deal of problems there, and the testing procedure wasn’t significantly different. I could still prove that everything worked if you tested these operands, but I got to know some of the guys who were working on Intel’s 80960. There were two processors at about that time. One was the Intel 860, which was designed by Les Cohen. Whether that was before or after the 386, I can’t remember. I think it was just a little bit after. The other was the Intel 960.

The 960 had a RISC architecture, and it had otherwise the same sort of floating-point capabilities that were built into the X87 architecture, but it was all on the chip of the Intel 960KB. Now, not all of the 960s had floating point. Some of them didn’t. But the 960KB had these floating-point capabilities, and I really liked them. I think they did a good job this time with the flat registers. No more stacks, thank you.

But something awful happened: the bean counters at Intel discovered that every chip, every 960 they sold, would be displacing an X86 chip that they might otherwise have sold; they were making more money from the X86 chips—lots more—and that therefore, the 960 really represented competition against Intel. To them, it didn't make business sense, and so Intel cancelled the 960. Well, they tried to cancel it. They cancelled it, only to discover that they couldn't.

You see, in the interim, a couple of things were happening: one of them should have been a joint venture with Siemens. Siemens was going to use the 960 in a workstation, and when Intel cancelled the chip, well, Siemens was left high and dry. I'm sure they weren't happy about that. That doesn't seem to have deterred the folks at Intel. They cancelled it anyway. But then, they found that they were selling a lot of 960s to people making printers, and they couldn't foreclose that market. That means they couldn't stop producing the 960s. Secondly, the Department of Defense standardized on two computer architectures. One was essentially a PDP-11 kind of architecture, and the other was the 960, so they couldn't close down the production line because it was a standard processor for the Department of Defense. So now they had the worst of all possible worlds. They had, in effect, tripped off the path for cultivating applications for the 960s as a replacement for the X86 architecture, but they couldn't stop producing the thing. And not just producing—they had to continue to develop it. I was very unhappy because, although my involvement with the 960 was little more than friendly acquaintance with some of the guys who were working on it, I thought it was a really nice architecture, and I regret its disappearance from the scene. That's how business decisions get made, isn't it?

The 860 was a monster of a different kind. The 860 was supposed to be a graphics processor, but Les Kohn used that as an excuse to design what he thought would be a "general purpose supercomputer on a chip." That was his phrase. It was intended to go on workstations, or even in a laptop. It had an enormous degree of concurrency without interlocks. That meant you could issue concurrent instructions, but there was no guarantee that the operands for instructions issued concurrently would actually have been ready from some other instructions. You were supposed to figure out the interlocks at the compile time. Furthermore, the arithmetic was a little bit cruddy. He had decided he wasn't really interested in all of this complicated Intel architecture arithmetic, and so he had a multiply-add operation. The multiply-add operation was designed to run faster, because he fetched three operands in the time that the others fetched two. And John Cocke had the same idea, in the Power architecture.

What Les Kohn did was, to save time, instead of rounding the multiply before doing the add, he would do the multiply, carrying a few extra bits, and then launch immediately into the add without rounding the multiply off. It would just carry a few extra bits and not bother about how it rounded it. Then it did the add. Then it did the round. So if you just want to do an add, you can multiply it by one and that will actually work, and then do an add. If you want to do a multiply, you just do the multiply and add zero. But for matrix operations, for instance, you do the multiply-add combined. Because of the way he did the multiply, it was possible for a situation to arise in which *a* times *b* plus *c* should be exactly identical to *x* times *y* plus *c*. Because *a* times *b* and *x* times *y* should have exactly the same values. However, because of the way he did the multiply, it was possible for *a* times *b* have its four bits to be different from *x* times *y* and its four bits. You see, *a* times *b* can equal *x* times *y*, even if *a* doesn't equal *x* and *b* doesn't equal *y*, and so on. It can happen. So that would mean that, under certain circumstances, you'd generate anomalies.

This is just the simplest way to describe the anomaly. The other anomalies were much more frightening, much more difficult to describe. And the Israeli who knew how all this worked had to go back to serve out his time in the Army, if my recollection is correct. So I think it may be that on the end, I am, perhaps, the only person in the world who still knows exactly what the 860 did—and it was perverse. Fortunately for me, unfortunately for at least two compiler companies, they went bust trying to write compilers that took care of the scheduling without the interlocks. And so we never did see compilers that could run the 860 at a significant fraction of its claimed peak speed. And so the 860 died. And I haven't shed a tear for the 860's passing, but I have shed tears for the compiler companies that went bust.

HAIGH: And did the 486 just take the exact same things that had been in the 387--?

KAHAN: Yes, almost. The 486 came in two versions. It was a 486DX and a 486SX. The 486SX was sold without a floating point for people who said they didn't want the floating point. However, the two chips were the same with one tiny modification. On the 486 without floating-point, all they did was disable the circuitry. That was it. If later you wanted to buy, I think it was called a 487, you changed your mind and you wanted to augment your crippled 486 with the floating point, what you did was plug this 487 into a socket; what that did was disable the 486 then, and it did all the stuff. It was a full 486DX, except it had the extra line to disable the other one. Again, this is marketing for you.

HAIGH: Yeah, I remember that. The SX, I think came several years after the original 486.

KAHAN: Not several years, very soon after.[12] I mean, there were always people who figured they would like to get a cheaper chip, you see. They don't need all this floating-point stuff, so they want to pay less. And so the Intel folks found a way of doing it, where it really wasn't changing anything much. I mean, it was just very minor changes in the way they put the pin connections to the chip. It sounds funny, I know; but that's the way the marketing minds worked.

HAIGH: Did the building of the floating-point units onto the main processors make a difference in terms of software support and the way in which the chips were used?

KAHAN: No, not that I know of. You see, by that time, hardly anybody with desktop workstations would get a chip without the floating point. In fact, there were some parts of the operating system, I think, that calculated the things about disks. I don't remember the details, but they used the floating-point instruction set. There were people who wanted to do only integer arithmetic. But integer arithmetic in that architecture was somewhat crippled, because a 32-bit-wide integer was all you got. And if they used the floating-point instruction set instead, they could get up to a 64-bit-wide integer, because that was the 10-byte format, you see. So there they are, doing floating-point arithmetic, when all they wanted to do was arithmetic with integers, because the floating point is wider.

So it really got to the point where almost nobody except people with certain special embedded applications…well, all your embedded applications of some kind didn't need floating-point—almost everybody else who thought they might need floating-point and didn't know whether they would or wouldn't, would get one with floating-point. It was cheap. And of course, Cyrix came out with the fast floating-point clone because, instead of using a digit-by-digit multiplier, they used an array multiplier. Instead of using bit-by-bit divide they were using a scheme that David

[12] Multiple web sources suggest that the original 80486 was launched in 1989 and the first 80486SX chip in 1991.

Matula had cooked up, which really is more or less predicted in [Donald] Knuth's book, where you think of the quotient digit as 16 bits wide, but you do it with a 16-bit register so you can have just a little bit of redundancy in your quotient-digit selection. It's was a very good scheme; they got a patent for it. They deserved it. It was a well-thought-out scheme. And they had transcendental functions and used an altogether different set of methods based on what are called Chebyshev polynomials; actually it's minimax approximation by polynomials. And that was done by Warren Ferguson, who had been a student of Matula's, and Warren did a very, very workman-like job.

The trouble came later. The trouble was that when the patent lawyers looked over the stuff, they saw that Warren was guaranteeing monotonicity and they didn't see anything quite like that in the obvious documentation elsewhere. So he thought, "Hey, this is a novelty—monotonicity. Let's patent it." Let's patent the concept of monotonicity, regardless of how you implement it, okay? Only a lawyer would think to do a dumb thing like that. So as soon as they made their patent reapplication, all hell broke loose at Intel and they said, "Hey, you can't do that, because our functions are monotonic, too. Kahan crafted them that way." And then we would be in violation of the patent. So there was this lawsuit and I became embroiled in it. It was really very tedious.

There were a number of patent claims, including patent claims about the stack and other things. I guess the unpleasantness of lawyers is something that I needn't go into here, because I would just be repeating what everybody knows: that lawyers can be extremely unpleasant. Well, at least the opposing sides' lawyers can be extremely unpleasant. You see, one's own lawyers are always exemplary gentlemen. Actually, the guys on my side and on Intel's side were. I supplied pointers to SHARE's microfiches of my submissions, of which some are in your file. Let's say the function is monotonic. I gave them a copy of this document with the monotonicity of some computer functions.

So you know, yeah, this is how we got it. We didn't have to do it the way the Cyrix people did it, and so we were able to defeat the claim for monotonicity. The other claims were still grinding around, but then the whole thing stopped—and Cyrix and Intel reached a settlement. Why? Well, Cyrix had counter-sued, because Intel salesmen had told customers that, if they bought the Cyrix chips, then their priority as customers for Intel chips might somewhat decrease. They might find themselves having to wait longer to get Intel chips. (And you know, Intel has been doing that again, or at least they're being accused of doing that again. I don't know whether they are actually doing it again. They did it then. That's sure. Whether they are doing it again is going to come out in some lawsuits). But because of that counter-suit, the Intel and the Cyrix folks reached an agreement. By then Cyrix had been bought up by IBM, so they could have had lots and lots of money. Ultimately IBM lost interest in Cyrix and sold it to National Semiconductor. So far as I know, Cyrix has disappeared from the scene. It's regrettable, because I think they did a good job and they deserved the good things they did. This lawsuit I blame not on my friends at Cyrix, but on the lawyers who didn't know any better.

HAIGH: As I recall, there was also something called the Weitek coprocessor.

KAHAN: Yes, Weitek coprocessors also could be used with the 8086…or maybe it was the 80286. The Weitek coprocessors only had add, subtract, multiply, and divide. They used an altogether different interface. It was, I think, called memory mapped. And it was faster, only for a while.

Oh, there were other coprocessors. There was one from IIT, which had truly bizarre arithmetic. I think some of it is mentioned in the notes about Cray. But IIT, I think, tried to cater to the graphics market by providing built-in matrix multiplications. It had some very bizarre aspects of its arithmetic. So I was happy to see it disappear from the scene. And maybe its other virtues just simply weren't adequate to keep it in the market. I just don't know. I have one of the chips, however, if you want to look at it. I mean, I actually plugged it in and used it in order to find out if it was idiosyncratic.

HAIGH: So I know you did other consulting work. We talked about the 8087. Perhaps it makes sense to talk about the IEEE floating point standards work.

KAHAN: Well, there was a little bit about the Pentium in there. The Pentium divide bug was partly my fault, too. Harsh Sharangpani, who was a novice about floating-point, came here quite often and sat with me right here, while we went through floating-point stuff so he could be brought up to date with what the 486 had, and then went ahead and participated in design in the floating-point for the first Pentiums. They were going to use an array multiplier, so they'd multiply real fast; that meant, in consequence, that the pseudo-multiply/pseudo-divide algorithm was not suitable. Therefore, he would have to design a special divide, and he thought an SRT divide would be okay. And I thought so, too, because it's a pretty standard design. The transcendental function codes were going to be written by Peter Tang, using the faster multiplies. And using also large tables, because a very important development was the precipitous decline in the prices of memory.

That meant that certain kinds of algorithms, which used large tables, could run faster than the algorithms that didn't use large tables and, instead, used somewhat more arithmetic operations. Now the idea was that you use either a high-degree polynomial (or a higher-degree rational function, or whatever) and a few table entries or you'd use lower-degree polynomials (with very few arithmetic operations to compute them) and very big tables. And there's a range of possibilities in between, so it's a design space.

As we move now to processors that are so fast compared with memory, it may be they can do the arithmetic faster than they can do the fetch from memory, we may be shifting back. We may go back to the older multiply algorithms with small tables, because tables take a while. On the other hand, if you have any tables at all, then you're going to fetch from memory a whole block of memory of contiguous cells. Maybe the fact that the memory is slow means that either you have no tables whatsoever or, if you have any tables at all, they might as well be big. I don't know. We're going to have to look into that.

There's also the question of power. Power dissipation is becoming a very severe constraint not merely on the portable machines but also on the desktop machines, because it's hellishly hard to get the heat out without having extremely noisy fans. Well, if you want to keep the power dissipation low, maybe we shouldn't use the bigger array multipliers anymore. Maybe we should go on with other things. So who knows what's going to come back; maybe pseudo-multiply/pseudo-divide will come back. We'll have to wait and see what the technology tells us.

At that time, though, Peter was going to use large tables because the memory speeds were not so disparate that that would be an unreasonable thing to do. Harsh designed an SRT divider and worked out a proof with Peter that it was correct. And then he called me, his old teacher, and said, "We'd be grateful if you could check on our design." And I said, "Well, you know it's a very standard thing. Do you really want to bring in the high-priced help to do that? I've got so many other things to do." So he said, "No"; I guess he felt that he and Peter could do it together. Well, this is the story now of hubris. The one thing I told him to do, in passing, is what he didn't do. When I said, "Remember to design the tests for your new divide algorithm, Peter knows how to do this"—but that didn't get communicated. If he had communicated it fully to Peter, I think Peter would have known exactly what I meant.

Anyway, they thought they had proved the algorithm correct. Actually, they hadn't. They had copied the ideas of the proof published by a fellow by the name of [Jan] Fandrianto. He published this in one of the IEEE proceedings on computer arithmetic. [J. Fandrianto, "*Algorithm for High Speed Shared Radix 4 Division and Radix-4 Square Root*," Proc. 8th IEEE Symposium on Computer Arithmetic, Como, Italy, pp. 73-79, May 1987]. But the paper, as he had submitted it, had a flaw. Jan Frandrianto had assumed that a certain array was going to work with assigned magnitude numbers. That means, in order to get the negative of a number, you just slip the sign bit. But, in actual fact, the way that array worked was twos complement, which means that when you want to take the negative of a number, you actually flip all the bits and then you add one. In consequence, his proof was incorrect.

And I remember that at this conference, I and a group of my graduate students and some other hangers-on had to get the guy literally cornered and point out that, you know, this is an oversight in his paper. And he finally said, "Gee, you're right. Maybe it was just that there was so many of us." In any event, he acknowledged it and inserted an extremely cryptic footnote in the paper as a correction. I don't think that Peter and Harsh noticed the footnote, or at least they didn't understand it. Their proof had the same mistake, but as it happens, the implementation of it was correct anyway, in that the table was right. It was just the proof that was wrong. Common, because proofs are a lot bigger than implementations sometimes.

So they did have a correct implementation, but in the course of transcribing the implementation to the taping of masks, something happened. I don't know whether a deck was dropped, or someone keypunched something wrong. Something happened, and the table that actually got laid out on the chip was not the table that they had specified. The table was a table of quotient look-ups. You look up a quotient digit in the table, and then that quotient digit is used to further the next step of the divide. Because of a certain amount of redundancy, your quotient-digit selection doesn't have to be perfect, because the next quotient-digit will compensate for any imperfections in the previous quotient-digit. And at the end, you reconcile all the stuff, because there's a certain redundancy in the quotient-digits which you get rid of by sweeping from right to left, which occurs very quickly. So this so-called SRT division process, named after the three guys: Sweeney, Robinson, and I think Tocher. This is all very standard, but the quotient selection table is essentially a trapezoidal table fitted into a rectangular box. That means that there are certain corners that never actually get accessed—they can't. In fact, the edges of the trapezium are accessed extremely rarely. It's the center of the table where most of the action occurs. And so as you go out from the center, the incidents—the frequency of reference—dwindles and dwindles and dwindles, until you get past this trapezoidal boundary. And then you never get there.

The entries along one of the boundaries got mashed. They got turned into "don't cares" when they should have been something else. And so I think they got to be zeros. Well, the algorithm is partially self-correcting, so under some circumstances, if you choose the wrong quotient-digit at this stage, the next quotient-digit can compensate for it, but not always. In consequence of that error, the quotient would be distinctly incorrect—in one case in nine billion or so trials with random numerators and denominators. One in nine billion. From one point of view, one could say, if that ever happens to you, we needn't worry about you. You're so unlucky, as Wilkinson would say, that you've been run over by a truck. In fact, that probabilistic argument was wrong. And the reason it was wrong was because it assumed that numerator and denominator were not only random, but uncorrelated. That's not the way things work.

In the meantime, some other guys were doing some investigation. Some people at IBM were investigating this bug when it became apparent and, from their reckoning, you would have a calamity every week, if not every day; therefore, IBM recommended that this chip not be used in any IBM PCs. And where was the truth? Well, of course, the truth was somewhere in between. When I was told that there was a bug in the divide, but I wasn't told what the bug was, I wrote a test program designed to test SRT dividers. I had some rough idea of what the dimensions of the table were, so knowing how many rows and columns there are in the table, or even knowing only roughly how many rows and columns there are in the table, it's possible to design a test which has a very high probability of detecting the bug, and it did. I think after some 400 trial divides, it found a bug. Others were using my divchecktest program. This was a program I had supplied to Cleve Moler when Moler was working for Intel, and they wondered whether the i860's divide was correctly rounded. They thought it was, but then they used my program for testing such things. It focused tests to see whether divide was correctly rounded. It was focusing at the right-hand end instead of the left-hand end of the quotient, so to speak, and it discovered very, very soon that no, it wasn't correctly rounded, which was somewhat distressing. But that was up in Oregon, and it didn't occur to the guys in Santa Clara to ask the folks in Oregon if they had anything to test divide, it didn't occur to the folks in Oregon to send it down, so the left hand didn't know what the right hand was doing. If they had run that they would have found, after some 4,000 test operands or so, that there was something wrong with their divide. You see, what they had done was to use the tests for the old bit-by-bit divide algorithm instead of writing new tests. The test should focus on what I might call singularities or potential singularities, and Peter knew how to do that. He had even written a little paper about it, but they just didn't do it.

HAIGH: So was the floating point for the Pentium was completely redesigned from the earlier chips?

KAHAN: Absolutely. Well, the add may have been very similar, but the multiply used an array multiplier taking a large amount of chip area instead of using a bit-by-bit multiplier, which just cycles and registers and shifts and, in consequence, the SRT divide had this table in it and other things instead of just doing some little cycling subtracts, tests, and shifts. So yes, the arithmetic was quite different—and Harsh was perfectly competent to design it. And he did! There was a little mistake, and they would have found it if they had only designed tests thinking about, "What are the weak points in this new algorithm?" And they would have found it, because I sure as hell found it quickly enough. Peter would have found it quickly enough. So I had to take some of the blame because I said, "You don't really want to high-priced help to waste their time on this; it's a routine thing," and Peter takes some of the blame because he didn't think to or Harsh didn't tell him to write special tests. Intel's brass takes a lot of the blame, because they believed a white paper in which an analysis showed that it would only be once in nine billion times and, therefore, wasn't worth worrying about. That's what they tried to tell people, and it was wrong because you've mistaken an algebraic function. It's not a random, probabilistic thing. It's a routine thing. It's going to happen every time you hit those operands, so the question is, how are the operands distributed? And because of the technicalities, they're distributed along the surfaces and broken like a kaleidoscope, so it looks like it's sort of jagged. But they're distributed in a regular, predictable way.

So Intel set aside finally $400-some million to replace chips. They finally agreed, after their arms were twisted by public relations, that if anyone had a chip with a defective divide and wanted to replace, they would replace it; they set aside money for the purpose, and I don't think they replaced as many as five percent of the Pentiums out there. Why, on the end, did it cost them so little? It was because the Pentium was upgradeable. It was in a socket—with rare exceptions. It was in a socket, and the intention was that when a faster Pentium came out that would fit in the same socket, you would replace it and you'd change a few jumpers on the board, and you'd be running now at a higher clock rate. Same bus rate, but a higher clock rate on the Pentium. That's an upgrade. And so Pentiums weren't being replaced, they were being upgraded.

Cleve Moler actually provided a program in MATLAB which, if you exercised it, would check to see if your divide was the kind that the Pentium screwed up and, if so, would do the job right. It would also record how often this happened. It would have been very interesting to find out how often this happened; it appears, however, that nobody outside of Cleve's immediate friends at Math Works ever substantiated it. Apparently they decided to take their chances, or else they were oblivious. I don't know. So the Pentium divide bug turned out to be a tempest in a teapot on the end; though there were all these folks that were doing something wrong, and even though people remember it painfully, as we should, it passed. No calamity was attributed to this bug, no great expenditure was incurred by this bug. Maybe a few million dollars, that's about it.

At the same time, roughly, somebody mentioned that there was a floating-integer store bug on the Pentium, but I have test for that. So I wrote a test program. When you store a floating-point number into an integer variable, which means you've got to round it off in some fashion to an integer and then store it: "Gee, has the job been done right?" So I wrote a program that would test these kinds of things. It discovered, yeah, there really was a floating-integer store bug, and the computer didn't like certain numbers…like 3/16. There was an almost identical bug on the Cyrix chip. I ran the same test on the Cyrix chip, and it just found almost the same set of numbers. Not exactly, but almost the same set of numbers were disliked by the Cyrix chip's floating-integer store.

Now, I know the guys in Richardson, Texas didn't ask the guys in Santa Clara or in Oregon how they designed a floating-integer store. Somehow, they had reproduced the same kind of bug. I wrote a little note about it. It's called "beastly numbers." The guys at Cyrix replaced their chip when I complained about it, and I did have a word with the folks at Intel. They had replaced it more than once, so I still have the processor. I'm going to get rid of it. I still have some that were on the old Intel Pentium with the floating-divide bug and the floating-integer store bug, but it's all past history. It's an interesting lesson in history, but I don't think it's worth keeping your computer around unless you think there's a museum that likes to exhibit computers with bugs in them, in which case, they would have to have a very large hangar.

So that's the story of the Pentium, other than the testing, which I think I've mentioned there. I did persuade the folks to test Peter's codes very thoroughly using the UBC test that Alex Liu had developed, and to run them fairly exhaustively; they did find a few spots where Peter's codes were a little bit less accurate than he had claimed, so he fixed them. I believe that, since then, nobody has found any bug in the transcendental functions part of the Pentium architecture. And that's one of the benefits of thorough and focused testing. It cost them a few man-years of effort because there were several people working on it, and they worked on it for several months, so it must have amounted to a few man-years. That's a cheap investment compared with what they would have to do if they had found some bug and had to go screwing around and retrofitting things and so on.

HAIGH: And how about the performance of the Intel floating point? I understand that, at the time the Pentium came out, people were expecting Intel to be swept away by these RISC chips and workstations but in the end, they seem to have rallied quite well and managed to improve their performance.

KAHAN: Well, because Intel had so large a share of the market, it was really selling a vast number of these chips at fairly high prices—much higher than the cost of production—so they had money rolling in and could afford to have literally hundreds of engineers working on the next generation of Intel chips. The other people, like the SPARC people and so on, didn't have that luxury. They really had to scramble for the money and for the engineering. The Pentium architecture is appalling. It's really bad. It's descended from the Intel X86 architecture, and perpetuates most of its faults. But the Pentium isn't exactly what's there. What's there is a little microengine which executes the most frequent instructions as any RISC chip would, in a very direct fashion and, as soon as the instructions get at all interesting, it goes to microcode. So how can you tell the difference? The RISC chips—they've got microcode too. Even the DEC Alpha, which must surely be the epitome of a RISC-chip design, had what they called PAL code, which was essentially microcode to handle exceptional or rare events. Ultimately, they used PAL code to cope with the penalties imposed by what were called trap barriers, but that's somebody else's history.

HAIGH: All right, so then the improvements in floating point weren't anything particular to do with improving the libraries or the things that you would be responsible for. It was more general improvements to the chip as a while and improving the materials technologies and the manufacturing processes and so on?

KAHAN: And maintaining compatibility because of these fast libraries which nobody wanted to reprogram. They had to reprogram them anyway, not because of the arithmetic, but because the memory management had to change.

HAIGH: Now, were all those MMX and SSE instructions that made their way into the processes, based on the existing floating point capabilities?

KAHAN: They were an attempt to respond to the demand for higher performance during multimedia activities, like graphical rendering and sound, music and sound, and so on. These applications can entertain vectorized arithmetic, which means that you perform the same operation on every pair of numbers in a column of pairs. Just take two columns, pair them up, and do these operations. That's very much what some of these computations are like, and so you really want to organize the pipeline in a rather different way; Intel was getting very good at putting lots of transistors on a chip, and so they gained space from transistors and put this stuff on the chip. That way, you could use Intel's chip instead of using a dedicated multimedia processing board with a dedicated chip on there. That hasn't stopped people from continuing to produce these boards—they just simply try to race ahead, economizing evermore on their processor's design. So they'll have a simpler processor which can run a faster cycle time, all on this auxiliary board that you plug in, and the only thing that limits it is how fast you can move things from the memory of your main processor onto the buffer memory of the board. There are folks who have even tried to get these boards to do floating-point operations like matrix multiplications for the sake of the main processor—offload the stuff for the main processor—but they found, so far that the interface is a terrible, terrible punishment.

Maybe that will change, but in the meantime, certainly, if you've got these multimedia instructions on your Intel Pentium or other Intel architectures, you will often find that doing the matrix multiply is somewhat faster that way. Of course, you don't accumulate the products quite as accurately, and that's discussed in my *Mindless* thing, where I explain that here you have iterative refinement which is sealed and set; you're accumulating your products extra-precisely on the stack, but somebody has goofed and blocked up these matrices so that the extra-precise accumulation is lost; then on the next generation of MATLAB, which is not discussed in the notes, they're using, as you call these, SSE and MMX instructions to do their raw matrix multiplies. So they can't accumulate the products extra-precisely, even if they wanted to. But it goes faster, and speed drives all common sense out of people's minds. If it goes faster, then it must be better. It doesn't really matter whether the result is better. Well, we'll attack that problem later. I think I've been told the name of the guy who codes the BLAS. I might be able to persuade him to get different versions of the BLAS that MATLAB will use under different circumstances.

HAIGH: As a consultant, you also helped IBM with issues with their RISC power architecture.

KAHAN: That's correct. This was actually being done simultaneously while I was still working with Intel, and ultimately even worked for a little while on the Itanium until Intel's lawyers presented me with one of these confidentiality clauses that went forever, and I said I wouldn't do it. So we stopped in 1995 or '96, which was sad. It meant that the Itanium, which is currently the world's most complicated computer except for the MTA (the Multi-Threaded Architecture) up in Terra, which is now Cray, in Seattle. That's the world's most complicated computer, and the second most complicated is the Itanium. Which is unfortunate. I think we could have simplified that, but the lawyers were adamant and I was adamant, and so we parted.

But yes, that overlaps a period when I went to Austin, Texas, called there because they had discovered, partly by running my divide check program (which was now in the public domain), and partly for other reasons, that it was extremely difficult to get divide correctly rounded using John Cocke's architecture. He was the one who actually originated the notion that subsequently came to be called RISC architecture in IBM. I would say that Seymour Cray had built the first RISC machine—the CDC 6600; at that time, RISC was in fad.

John Cocke wanted to do that in 1967, remember, when I visited IBM; there I was, walking down the hall, and this long, bony arm reached out from Cocke's office, grabbed me by the collar, and pulled me in. He said, "Kahan, I want to show you something," and he was showing me the origins of what subsequently came to be known as RISC. But when he did that in 1967, compilers simply weren't clever enough to do what he wanted. By 1984, compilers were clever enough. Cocke wanted to simplify the instruction set drastically, so instead of having add, subtract, multiply, and divide, all he had was a multiply-add, and from a multiply-add, with minor variations, you could get multiply/subtract. You could then turn that into an add/subtract. If you multiplied an operand by one, you could turn it into an array multiply if you added zero--you've heard about this already, because I told you about it with the i860.

And they discovered, as the folks at Intel had discovered, it was awfully hard to get divide correctly rounded to meet the IEEE specification if you did it that way. So they thought they'd call me down in order to see whether there was anything I could do about it. Maybe I could change the IEEE standard for them. But, as I was looking over the chip, I said, “You know, you’ve got some space here. If you can do what is now called a *fused multiply-add*”; I meant, “Do the multiply exactly, then do the add, then do a rounding error. And if you can do that,” I said, “You can computer remainder, and if you can do that, you can figure out fairly easily how to get the rounding correctly.” In fact, there were guys at Yorktown Heights who were doing things like that—Brian Tuckerman—and you didn’t even have to do it his way. It really wasn’t all that hard. And then you’d get a correctly rounded divide at a perfectly tolerable cost under the circumstances. And incidentally, this thing would be handy for things like decimal-binary conversion and for other stuff, so it wouldn’t be used if it was just a one-off.

I thought (and continued to think, until about a month and a half ago) that this was my idea. They implemented it and it worked, and then they had to ask, “Does this conform to the standard?” and I would say, “No, it’s an extension beyond the standard but, actually, it’s been written into the standard now.” But I discovered that the guy who did this, his name was Robert Montoye, I thought that he was really quick off the mark. As soon as I had suggested it, it was as if he took it into his garage one weekend and came out and it was done. What I learned a month and a half ago was that he had investigated such things in his thesis, and the possibility now exists that, in fact, he had wanted to do this all along, but what he really needed was somebody was to come in with a reputation or whatever it took to persuade folks around him to let him do it. As you know, IBM doesn’t really like the idea of doing things which, later, it’s going to have to support in all its subsequent machines; they weren’t too happy to have this innovation in there. Then I came along and came up with it—I thought, myself, but apparently it was in the air. I was just filling in a blank. That’s why Montoya did it so quickly. It could be. I’ve asked some friends who know Montoya to find out what actually happened. It would be nice to get the history straight, you see. It’s possible, then, that Montoya had this idea all along. It’s possible that he was just waiting for someone to let him do it. It’s possible that I turned up, persuaded folks that this would be the right way to solve some of their troubles, and it was.

And Peter Markstein—I’d met him originally in Yorktown Heights—he was now in Austin working for IBM on this particular project. He found lots and lots of ways to use the fused multiply-add to advantage, so it turned out to be a great thing for them. Then, when Apple was thrown into bed with IBM by the Pepsi-Cola man, Scully, Apple adopted the same architecture. Motorola got a license to produce the chips, initially. So Motorola was very disappointed to lose Apple as a customer for their 68000 family, and also for their subsequent RISC family. That was the 88110.

The 88110 was a Motorola chip with a RISC architecture and beautiful floating point, just like the floating point on the 68040. The 68040 did thoroughly what John Palmer had wanted to do, but didn’t have enough transistors to do. They did a really nice job. Mind you, they made one mistake, and it was a funny mistake. See, they used a CORDIC algorithm (remember, I said there’s CORDIC, which is part of a family of algorithms). I was using pseudo-multiply, pseudo-divide. They were using a CORDIC algorithm, and the CORDIC algorithm isn’t quite so accurate as pseudo-multiply, pseudo-divide, but on the other hand, the CORDIC algorithm can run faster. It can go some 30% faster.

The CORDIC algorithm was written up really well by Steve Walther. He worked at IBM in the Deer Creek Road research facility. He wrote a paper on CORDIC algorithms for computing transcendental functions, not merely trig functions—hyperbolic functions and, consequently, log, exponential, things like that—and he presented the algorithms in APL language, so he really had everything you needed.

But he left one thing out of his paper: how many extra digits must you carry in order to get a certain number of digits right? And the answer wasn't perfectly obvious, but when the guys at Motorola were in the throes of implementing this algorithm, they were calling Steve Walther for advice whenever they came to a sticking point, and Steve Walther, being a good-natured fellow, was giving them advice until his management at Hewlett-Packard said, "What are you doing giving away this kind of consulting information to a rival or at least potential rival?" So the next time they called, Steve had to say, "I'm sorry. My boss has told me I shouldn't talk with you guys."

What were they calling him about? They wanted to know how many extra digits they should carry in order to get 64 correct. Well, he couldn't tell them, and they didn't figure it out quite right so, in fact, they were losing a few bits of the 64. On the initial coprocessors for the Motorola 68000 family (that's the 68881 and the 68882, slightly faster), they were losing bits, so their transcendental functions were not so accurate as Intel's and, indeed, therefore not so accurate as Cyrix's. But they were pretty fast, and they had built the complete library on the chip, doing what John Palmer had only hoped to do and, ultimately, they redesigned things so that they could put the floating point on the same chip as the processor and not need a coprocessor, and that was the 68040. For the 68040, they employed Peter Tang as a consultant. He wrote a beautiful library for them in which they got all the accuracy they needed. And I have a 68040 and I'm delighted with it. It runs in my Macintosh Quadra. It does a beautiful job but, unfortunately, it does it with a 33 MHz clock.

Well, it looked as though the RISC architecture fad would be inappropriate for the 68000 because it was a CISC architecture, if ever you saw one, a complicated instruction set architecture with an orthogonal instruction set and everything else. So one could have said, "Well, you've got a slow machine," but Motorola had designed a RISC family of machines which had the same floating point as the 68040, just implemented a little bit differently. It used the same transcendental function codes, just implemented a little bit differently, and Apple rejected the use of that processor because Scully wanted Apple and IBM to become buddies. So that killed the 88110, which was a good architecture. I think I like the 960 better, but the 88110 was a pretty good architecture. I've got the manual here, even though I haven't had the chance to look at it carefully, and that screwed things up at Apple.

Apple had an excellent numerical environment called SANE, Standard Apple Numerical Environment. It had so many of the things that I had wanted. They didn't necessarily do it in quite the way I liked, but they did a very conscientious job, and programmers came to like it a lot. Of course, it takes a while. There's a lag. You know, initially the programmers are complaining about everything. They'll complain about any change, they'll complain about having to spend extra time to get the right answer and stuff like that, but ultimately, the programmers were really liking it. These guys were getting unsolicited testimonials from programmers and some letters to the editor and so on about what a great thing that SANE was, and that was killed in order to switch to the new Power PC architecture.

HAIGH: So the thing you consulted on, that would have been the original Power architecture--?

KAHAN: Correct, RS/6000.

HAIGH: And that was implemented in four chips, wasn't it? And then the Power PC was a kind of miniaturized subset?

KAHAN: Yeah, the RS/6000 had a cluster of chips initially, but nobody doubted that it would ultimately get all onto one chip; the fact that it was four chips, that really doesn't matter a hell of a lot.

HAIGH: So that fused multiply-add became a standard part of the Power PC too, did it?

KAHAN: Correct. And Apple used the same kind of software. Not necessarily the same software, but the same kind of software. The fused multiply-add made it feasible to implement something called *Doubled Double*. I know it's an adjective, but we're using it as a noun. Okay. And that was a sort of grotty substitute for quadruple precision; it had been programmed by Peter Markstein and his wife for the Power architecture; similar code was produced within Apple for the Apple version of the architecture. That was one of the advantages of fused multiply-add, that it allowed you to implement this somewhat grotty approximation of quadruple precision and run it fairly fast. So that is the good thing that they had in order to substitute for the good things that they had taken out from SANE, because the Power architecture couldn't support SANE the way it was.

I mean, to give you one example of what was lost: SANE had an integer type called *comp*—computational type—intended actually for COBOL, among other things, but it was a 64-bit-wide integer and then a sign. It was very hard to get that from the ordinary integer instruction set of processors at the time. They were limited to 32-bit-wide integers. But here was a 64-bit-wide integer which you could use for computation; if an overflow occurred, as happens with integer computations from time to time because it was done in the floating-point register, it would get rounded, and you might discover that all of your numbers are multiples of eight all of the sudden. That's rather strange, but there was an inexact flag that you could use, and there was an integer NaN, which meant that, for some occasions when you compute with an integer, you have to return an integer because the destination is going to be an integer register no matter what you do. Instead of having to stuff some unfamiliar integer in there, hoping that somebody would notice that, "Geez, this is a weird integer answer for this computation," you could stick in an integer NaN. It was a floating-point NaN, and so it changed the complexion of integer computation when the integer is the data rather than merely the indices. SANE had a lot of really good things. Good exception handling. Not the way I would like it, but it was good enough. And it was killed, and that set us back for some considerable period of time.

[Tape 10, Side A]

KAHAN: The IBM thing is jumping ahead because the IEEE standards stuff came before that.

HAIGH: Yes. So just to wrap up this discussion of your consulting work, clearly you have been involved in lot of different projects and they have been important assignments that have had a considerable effect into what's happening in the world. How do you distinguish the kinds of satisfactions and challenges that you face in the consulting work from the kinds of satisfactions and challenges that you face in your academic research?

KAHAN: The biggest challenge in my academic work was finding students who were inclined to this kind of work. It's very hard to get good ones. The consulting work, at times, paid me more than the university could or did. I wasn't doing it for the money. I was obviously doing it for the opportunity to, as you say, to influence events. There was this work on the WorkSlate, which was I believe the first laptop, and it was aimed at business computation. It was done in decimal and had nothing to do with the IEEE standard, but it did have to do with was getting some very reliable functions into a world accustomed—alas—to unreliable functions. So I have to say that the arithmetic was superb and the business functions, like calculating interest payments and stuff like that, was punctilious. However, when I saw the keyboard, I groaned. It was a really unpleasant keyboard. Overall, the whole thing was ugly, although it had a telephone modem in it and all sorts of nifty things, but it was an ugly device. It fizzled in the marketplace for reasons having nothing to do with the arithmetic because I don't think anyone ever got that far looking at the product.

There were a few other consulting jobs, I can't remember. My only concerns were to avoid conflicts of interest. I didn't want to work for two people who were going to go after the same market. I didn't want to consult in a way that would conflict with my university obligations. On occasionally, I would actually take a leave of absence called "industrial leave." I didn't want to sign confidentiality agreements that would obstruct my ability as a professor to expose secrets rather than hide them. The tasks were chosen because of the opportunity to influence events, sometimes, because I would end up with more students. It just happened to be in industry instead of in the university classrooms.

I did not use the university as a base as some of my colleagues did. In any event, even when I was on industrial leave, I continued to work with my graduate students and come into the office and attend to whatever students thought they had some reason to see me. It might be a graduate student asking for advice about thesis problems and computations, students coming in for advice, generally. Because of my very peculiar hours, which would sometimes have me working late at night, students would come by who just wanted somebody to talk to. I don't know why I seemed so avuncular to unburden themselves to me, because normally students would be somewhat frightened of me. Here there were students who wanted somebody to talk to so I would try not to disappoint Sheila. I would patiently listen to them, things like that. I don't think my consulting ever interfered with the university work. I think what it did was to complement it, and in some cases promote it. But what is true is that neither the consulting nor the IEEE standard stuff was what I would have wanted to do if the world had only been set right arithmetically. I wanted to solve certain kinds of differential equations and to perform certain kinds of matrix computations involved in solving those differential equations. I was interested in a few other computations, too, but these are the ones, which I felt I was best suited.

HAIGH: So it would seem appropriate now to turn our attention to the IEEE standards ethics for floating-point hardware. Now I should say for the benefit of readers of the transcript at this point, that we have tracked down a longer version of an interview conducted on this very topic with Charles Severance. A shorter version of this was published in the March 1998 version of *IEEE Computer*, but there is currently a longer version to be found on the Web, which summarizes the general history of the committee with particular respect to battles over gradual underflow and the dynamics of that between the KCS proposal and the proposal being put forth by DEC.

So, returning to the origins of it. I think you had said earlier that you, personally, as you listened to the discussions in the early and mid-1970s that people were having about how to make mathematical software more portable, became convinced that the best and only really satisfactory way of doing it would be to standardize the arithmetic itself rather than to work around--

KAHAN: It wasn't standardizing in the sense of the IEEE; it was rather a prescriptive formulation of arithmetic rather than descriptive. At that time, I was willing to contemplate the possibility that there might be three or four preponderant schemes, which had become *de facto* standards. I could have lived with DEC's arithmetic; if IBM had decent arithmetic it would be different, almost surely. But if a couple or three major manufacturers—Hewlett-Packard and DEC and IBM—had their own arithmetics and arithmetics of different parameters and maybe different word sizes (though that is unlikely), I could live with that, provided that the arithmetics were prescribed in the sense that, once you knew the word size and the precision and the exponent stuff, you knew everything arithmetic was going to do. If you could do it that way, then we could write portable software and there would be only a few constants that you would have to slip in to adapt the software to any one of these comparatively few different arithmetics. I did not contemplate even when working for Intel initially that there would be one standard for everybody; there is no Pope: who is going to enforce it? I didn't know.

Then somebody at Intel must have boasted about this marvelous thing that Intel is going to put in, and it's going to all go on one chip. As I said to Severance, "I suspect that the impetus to join this enterprise that Bob Stewart started was to see whether Intel could be slowed down." That is my suspicion although I could never prove anything like that.

HAIGH: So who is or was Robert Stewart?

KAHAN: R. G. Stewart was a guy who was running a sort of consulting outfit out of his garage. He is still alive so as far as I know. He had been active in IEEE activities and as an applications programmer and as a consultant in these affairs. And he was distressed because, at the time, it appeared that every manufacturer of microprocessors would be going off in different directions. And these manufacturers are springing up like mushrooms. It was the beginning of the sort of dot-com expansion era, except it was microprocessors instead of just the communications—people were building things rather than ideas. But everybody was building them, and everybody fancied themselves a computer architect. Every computer architect was ignorant about floating-point, and they were going to do it in whatever way struck their fancy. So Bob put together this committee. He had much higher ambition because he wanted to have a standardized assembly language and lots of folks thought that could happen. It seems to me to have been a pipe dream, but there is a more-or-less standardized assembly language now. The thought of standardizing floating-point arithmetic, I think, should have been a pipe dream, too, except that there were a couple of designs that were candidates. DEC designed for the VAX was actually a very good design, considering what we knew in the early 1970s; it came out in the mid- or late-1970s. As designs of that era go, that was a pretty good design—certainly, when the decision to buy a VAX was being brooded and I looked at the floating point and said I can live with this a hell of a lot easier than I get live with the CDC 6400 or any other machine.

HAIGH: I have a question about your attitude towards that. It seems that what you are saying now is the important thing is to have a standard and that standard be something that people can live with. But your article "Why We Need a Floating-point Standard" doesn't talk so much why a standard is a good thing in itself; it talks more about how dreadful many of the existing things are, and how we need to get rid of them all and install the best arithmetic we can possibly have.

KAHAN: By then, you see, my ambition notched up a bit. First of all, much to my surprise, Intel was building this thing and I then realized that there was a chance of having a very small number of dominant arithmetics. Now I didn't think that DEC would disappear from the picture, and I think it's fair to say that it wasn't the arithmetic that pushed them out of the picture. I didn't think IBM would disappear, and I don't think IBM has disappeared yet. It doesn't look as if it will. So there is this abominable hexadecimal arithmetic that will continue to soldier on. But I was fairly confident that Intel would be a very significant player. That's because, first of all, they were producing an awfully large number of the 8080s and, secondly, they were producing a lot of memory. Third, it was customary in the industry to second source, so whatever it was Intel was going to build was going to be built by at least one another guy. AMD was the logical choice, because there were already a substantial number of second-source agreements between Intel and AMD. Second-source was necessary because very few people who built larger assemblages were willing to stake their future on a single source.

So whatever Intel was going to do, some other major manufacturer might do. There was a tendency for people to copy things. I knew that Intel could not and would not patent the format for the floating-point numbers, because the ideas for that were in the public domain. Maybe the parameters exactly weren't, but the ideas were in the public domain. And all I was really doing was choosing, eclectically, the best things that I could find…and inventing very, very little. I invented the way NaNs should work, instead of doing it the way Cray did it (the German fellow, Konrad Zuse, had similar ideas sort of about an indefinite quantity). Cray had built a more successful indefinite in the CDC 6600, but that software hadn't supported it, and hardware didn't either, really, so he more or less abandoned the distinction between infinity and indefinite when he built the Cray machine. I figured out that I was going to get that right, so there was a certain amount of invention there. But not a hell of a lot. If someone tried to patent that, I think I could have undermined the patent application.

There was the inexact flag—I certainly invented that. That was something that I knew about because of J. C. P. Miller. Miller knew that there were times when you could notice whether something was exact or not, and that made a lot of difference to a computation. The old guys did a great deal of that, and I said no, I can see what to use the inexact for and I can even see how to justify the inexact for others who aren't going to use it the way I would, namely the guys who have to do big integer arithmetic. So, yes, I invented the inexact flag.

Everything else was deduced. Everything! It was an eclectic mixture of the best that I could find. I deduced the necessity for the extended format from the experience we'd had with the calculators and from the experience before that on the 7090. I knew we needed an extended format to support double precision in systems like MATLAB and MATCAD, which used to be called turnkey systems. I knew it had to be there and I knew it had to be specified in a way that would allow it to grow—that if we needed higher precision, and I expected we would, I thought someone could take the extended format and add more precision to it. So instead of 10 bytes, it would go to 12 bytes, depending on how long it takes to do the carry propagation and whether it fits into the cycle window. So a lot of that was just deduced and, in consequence, I knew it had to be done that way, that really it's like an engineering job where finally you figure out why everything has to be the way it is.

To my recollection, the inventions were inexact flag and the behavior of NaNs. The way NaNs behaved had to be constricted because NaNs didn't exist in the languages and therefore, for example, a certain kind of abomination, NaN had to be unequal to itself.

Signed zero—well, the signed zero was a pain in the ass that we could eliminate if we used the projective mode. If there was just one infinity and one zero you could do just fine; then you didn't care about the sign of zero and you didn't care about the sign of infinity. But if, on the other hand, you insisted on what I would have regarded as the lesser choice of two infinities, then you are going to end up with two signed zeros. There really wasn't a way around that and you were stuck with it.

Gradual underflow was something that you were stuck with it because you knew that underflow was going to be ignored. I've written about that in my short tutorial on underflow. You know it's going to be ignored no matter what you do. What has the best chance of being ignorable under a reasonably wide range of circumstances? Well, that was gradual underflow. I know that I had implemented it on a 7090 before a paper by I. B. Goldberg came out. But I think we were calling it the Goldberg Variation, so we must have known something about that. I don't know exactly how that came about. I can't remember exactly and whether I created it out of whole cloth or whether I got wind of the Goldberg thing…I just can't remember anymore. But I know that I implemented gradual underflow at least a couple of years, if not more, before the Goldberg paper actually came out [I.B. Goldberg, *27 bits are not enough for 8-bit Accuracy*, Communications of the ACM, 10 (1967), 105-108]. And I don't take credit for it; I think it was just so natural and so intrinsic in the system that it didn't have to be invented—it was already there. It was there in the un-normalized instruction set, so to speak. Just use the un-normal instruction set in a reasonable way and it was already there, so I can't take credit for inventing that.

Well then, what's left? You've got to choose the balance on the exponent range, and if we are going to have gradual underflow than overflow is the more serious threat so you want the overflow threshold to be a little bit higher than the underflow threshold is low. It all works out that way. That's the way in which the design proceeded—by deduction, rather than by mere inspiration and creation. I think that that pattern was evident to the people who took the trouble to look, and that's why they accreted to the standard. When you looked at it, it had this integrity and there was a certain connectedness about it. Things seemed to make sense to those who looked at this. And of course, you could only look at it through eyes that had experience with a certain number of computers before, or where you knew you had been bitten by one bug or another.

HAIGH: Now, as your own attitude shifted from the idea that a reasonably good standard would be good enough to the idea that you really wanted this best standard that you and, as far as you could see, anybody knew how to make, did that cause problems for the dynamics with the other committee members? Were there people on the committee who still had something close to your original attitude?

KAHAN: Sure. I think the DEC folks had an attitude like that at times. They had a certain ambivalence, because they would have been really, really happy if there had been only one standard that had been DEC's. And it wouldn't have been an unreasonable thought, but Mary Payne, who was right about many things, brought on some guy who I came to dislike, and so I've forgotten his name. He said, flat out that it couldn't be done. I can vaguely remember his name and I want to forget it. He was a disagreeable character, and he said things which he could not subsequently back up. And unfortunately, Mary trusted him too much at first. That's why she lost her credibility in the committee, I think undeservedly, because she was…you could call her rigid; you could call her old-fashioned; but she had a certain integrity, and she was competent.

George Taylor did it to show that it could be done—what's more, he did it twice; he did it for the board from the VAX Unfortunately, there was a dreadful swimming accident, so the guy who was supposed to test it died. He hit his head on the bottom of the swimming pool and nobody noticed for too long.

HAIGH: Did he have the name David Cary?

KAHAN: That’s right, Cary. He was the one who was supposed to run the tests after George had designed the board and the layout and chosen all the components and everything. David Kerry was going to run the test—not single-handed; he had some help. Gloom descended upon us with his death, and I know it affected his family badly; I suppose it’s not a hell of a lot of comfort to them to know that it depressed the rest of us, too. That’s all I can say about how that came about.

So, nonetheless, it was perfectly clear that George’s thing was going to work. That was the trouble: you didn’t really have to test it; you just had to look at it to see that this was going to work. Then he did it again for the Elxsi 6400, using now a different circuit technology—ECL instead of TTL—so he really got it to go fast. How can you argue against that? And the 6400 was a successful machine. They sold a lot of them, and if they hadn’t gone nuts with their second design and destroyed the company with fractious contumely within the company, they might be with us still. Of course they would have had to abandon ECL; it was just too expensive. Sure, Intel can do it, because they can throw vast numbers of engineers. But once we knew that it was feasible by others than Intel, once we knew this could be done by a graduate student (a rather special graduate student—George wasn’t a run-of-the mill graduate student; he was very special) how could you say it was impossible? This undercut the credibility of DEC’s arguments, even when their arguments had good substance to them. This was most unfortunate and I would really have preferred to have the discussion at the level of technical discourse and education. Alas, it’s a committee and, as I’ve told you already, committees have a dynamic of their own which is not altogether admirable.

HAIGH: In October 1979, the *SIGNUM Newsletter* published a draft of the standard--

KAHAN: That’s right.

HAIGH: Which was credited to Coonen et al.

KAHAN: Yes.

HAIGH: How much had changed from the original KCS draft that was presented to the committee and this draft from ’79?

KAHAN: Very little. Draft ten, I think, was the final draft. What draft is that seven?

HAIGH: 5.11.

KAHAN: Oh right. There were changes. I can remember a number of the changes, and one of the changes that really, really hurt me was the rejection of the projective mode, the distinction between projective and affine mode. Coonen, whose political sense I think is better than mine, put this forward as a way of garnering more political support and, if I had had my way, I would not have tolerated that compromise. It was the wrong choice. If you’re going to choose only one infinity mode, there should be only one infinity and only one zero. So I lost that one. And there is a mistake in the standard, in consequence. If you look at the predicate—*x* less than or equal to infinity—that’s a predicate that’s true for every finite number and every infinite number; therefore, it really ought to be true for NaNs. But, unfortunately it’s false. In the projective mode it could be false, that would make sense. But in the affine mode, which is the only one the standard selected, then that turns out to be a bug.

HAIGH: Let’s go back and talk a little bit more about the early days of the committee. You said that Robert Stewart had initiated it.

KAHAN: Yes.

HAIGH: The first chairman was someone called Richard…

KAHAN: Richard Delp.

HAIGH: Who was he?

KAHAN: I think he was at Four Phase.

HAIGH: Was that a semiconductor--?

KAHAN: It was a company; I don't even remember what they did anymore. They used to have a building, they don't exist anymore. Richard Delp is retired and we see him from time to time, maybe once a month—he turns up at the monthly meetings of the present committee. Delp was not really the best guy to be a chairman. He didn't have quite the drive and focus that Dave Stephenson had. Dave Stephenson succeeded him.

HAIGH: These early meetings would it just have been that anyone who wanted to could turn up?

KAHAN: Absolutely, and did.

HAIGH: These were mostly the people from microprocessor firms?

KAHAN: That's right, yes. Most of them were going to be building microprocessors and, boy, it seemed like an awful lot of guys were going to be building microprocessors. They came. One of the reasons they came was because the sessions would get educational. These sessions would sometimes run as late as two in the morning, discussing how to actually do things, and why some obstruction isn't really an obstruction. Somebody says "How do you do such-and-such," and somebody else would say, "You do it this way."

HAIGH: How did the character of the meetings change over time as it became more of a formal committee?

KAHAN: I don't know if it ever did become more of a formal committee. We had to put the draft out to vote. We had to deal with each negative vote.

HAIGH: So who voted, then?

KAHAN: If you had attended a sufficient number of meetings, you got to vote. IEEE changed its rules afterwards. Our rules were after you attended your second meeting, you had a vote. I think it was as simple as that.

HAIGH: So it remained the case that pretty much whoever wanted to turn up could do?

KAHAN: Oh, DEC tried to pack the meeting once, brought a whole bunch of DEC guys down. But we heard they were coming, so I brought a bunch of graduate students and some others, I can't remember. Packing the meeting turned out not to be such a hot idea.

HAIGH: After a while, the effort became called P754. Presumably IEEE just allocated that number as it's next free number for a standard.

KAHAN: Yes, I think so.

HAIGH: So what do you have to do to become an official numbered-- ?

KAHAN: Boy, are you asking the wrong guy! I don't know. That was Bob Stewart's domain; that was the domain for the chairman. They were the guys that had to figure all that out and I'm grateful that I didn't have to think of it a bit.

HAIGH: Can you remember was it called P754 for as long as you can remember, or did it used to have a different name?

KAHAN: No, it had P754 for as long as I can remember.

HAIGH: So in the very early days, people already had the idea what they wanted this to lead to was an official IEEE Standard.

KAHAN: I think what precipitated it was the fact that we had a draft. It's like the old Civil War general who says, "He wins the battles who gets there firstest with the mostest." Coming up with the draft as we did, we got there firstest with the mostest. I didn't realize how important that was, until afterwards.

HAIGH: Looking at this published draft standard, I think there is a list of people who contributed to it. Here it is: Jerome Coonen, William Kahan, John Palmer from Intel, Tom Pittman--

KAHAN: Yes, Tom Pittman was really a compiler guy, and he specialized in writing compilers for diverse microprocessors.

HAIGH: Did he help by producing some kind of sample compiler design that would work--?

KAHAN: No, I think what he did was to form his own judgment about whether this stuff could be compiled to, and I think he lent moral support; he saw that there was the mathematical integrity in the arithmetic that made more sense to him than the floating-point arithmetics that he had had to look at other machines. Because the floating-point arithmetic on most of the microprocessors were all software simulations of something.

HAIGH: You know, I think I may be answering my own questions to some extent here, because it all gives an address to send comments to. The address is "Secretary of the Floating Point Working Group." So do you think that Floating Point Working Group would also have been a name attached to this effort?

KAHAN: I don't even remember what the name was, anymore. It was a committee of the Microprocessors Standards Committee of the IEEE, or something like that. If somebody called it the Secretary of the Floating Point Working Group, well, that's fine.

HAIGH: Perhaps this would be an appropriate time to talk a little about [Jerome] Coonen.

KAHAN: Coonen? Coonen was one of my three best students. For a while one of my two best students. But all very different. One was Jim Demmel, whose mathematical competency and energy was just phenomenal. He wrote an absolutely spectacular thesis. Coonen was competent enough mathematically, but Coonen had a certain sanity about him and he was much better at communicating the things he understood than practically anybody else on our team. He had a good sense about what was bothering people. He was quick on the uptake for things like that. Because Coonen was sane, he decided not to continue working as he had been—I think as an employee for some time at Zilog. He was the mathematical consultant for the design of the Zilog Z8070 coprocessor which I think was-- well, some were actually built, but Exxon bought the Zilog and then tried to convert it directly into a profit center, and it killed the company. They killed the Z8000 and the Z8070 which, would if it had continued to develop, had been the fastest of its time. It would have been faster than Intel's and even faster than Weitek. It looked like a good architecture—not, maybe, a superb architecture, but a pretty good architecture for a microprocessor. But there is a Z8000 residue or fragment, I wouldn't know what to call it, maybe fossil would be the right word, which still exists—development from the Z80. But they never became a significant force, unfortunately. So Coonen worked for them; I know he worked for Apple for some time.

Then Coonen decided, and this is what I realized later is what distinguishes him from the rest of us. He decided he wanted to be present to watch his boys grow up. That he could earn an adequate writer as a technical writer. So instead of become an employee at various outfits, he became a contractor who would write manuals if they were sufficiently intricate or technical or mathematical or whatever. And his wife earned a living in a similar sort of way.

HAIGH: Was all this was before he entered graduate school?

KAHAN: No, this was after.

HAIGH: I see he graduated in 1984.

KAHAN: That's about right. I don't know; I don't remember the exact date. I'd have to look at his thesis.

HAIGH: So all of this would have been after his work on the committee?

KAHAN: Yes. His work on the committee got extracted and summarized in his thesis where he provided the justification for some of the design decisions, like the decimal binary conversion decisions and some of the others. Indeed there were so many of them, that if he had written done all of it his thesis would have been four or five times as thick and it really wasn't necessary. His thesis was a superb computer science thesis, as it happens, written in a math department. But the rest of the committee (of course I was jointly appointed), but the rest of the committee thought it was fine. Whether they thought it was fine because I thought it was fine, or whether they thought it was fine on their own hook, I really cannot say.

Coonen had started out interested in solving partial differential equations and he would have worked with one of colleagues, Ole Hald. But one day Coonen came in and asked me why it was that the PDP-11 did this strange thing in a course of program. And you can never get a short explanation from me, so one thing led to another; the next thing he knew, he was trapped in floating-point. I invited him to help with this standard when I saw that he was articulate and modest. He would not claim anything that he did not have good reason to believe. I was really glad that he accreted to the project. And, of course, we went to get Harold Stone because the problem was we didn't know exactly what a standard look like when you draft it. We knew what we wanted to say, but how to you say it in such a fashion that it would look like standards and that was Harold Stone. And what's more, Harold liked it. He liked to be part of that, it looked like a good idea to him.

HAIGH: I know he was visiting, but what was his background?

KAHAN: Harold's background had become computer architecture, really as sort of a commentator, perhaps, rather than as an assiduous builder. I don't remember what he did before that but I know he did some programming and numerical computation. Not a great deal. He certainly didn't think of himself, nor did we think of him, as a numerical analyst or anything like that, but neither was he ignorant of. But you've got to remember, in the early days, when people were into computing, they were into computing doing all sorts of things. After all, *computing* meant initially *numerical computing*.

HAIGH: How many meetings did this group have before you presented it with the KCS draft? Had it been going on for, say, two years, or was it relatively new?

KAHAN: Oh no, I presented the first KCS draft in…it couldn't have been any later than 1979. It was more like in 1977. Coonen may actually have all the old drafts in his file cabinets in his basement somewhere.

HAIGH: So the date we have from the Severance interview is that the second meeting (which was the first one you attended) was in November 1977. So if that is correct then that would mean two years later the draft--

KAHAN: But that's draft five.

HAIGH: Yes, draft five. The other documents were published in the SIGNUM newsletter.

KAHAN: I guess we must have submitted our draft somewhere in mid-1978, I don't know exactly.

HAIGH: So it seems the committee was working fairly fast as such things go.

KAHAN: Light-speed in comparison with other committees I'd seen. But that was it, we got there firstest with the mostest. We got there with a draft and DEC got there, of course, with their proposal, which was the manual for the DEC VAX.

HAIGH: Can you talk a little bit more about the contributions that each of the three of you made to the first draft?

KAHAN: You mean, me and Coonen and Stone?

HAIGH: Yes, it's got three people's names on it so presumably each of you--

KAHAN: Well, I told you Stone was an enthusiastic participant helping us shape this thing into a form that would like a standard. He didn't pretend to be a numerical expert. But insofar as he could understand what we wanted to do, his experience told him this would be a good idea. So he accreted, if you like, to the cause.

HAIGH: So he made it look like a standard?

KAHAN: Yes, that's right. He feigned it that way. Now Coonen was the one actually drafting it, and I was the one whose ideas were being explained, for the most part, although Coonen modified them a little bit from time to time to make them clearer, or sort out what might have been ambiguities or even dissonances. I had a pretty clear idea of what I wanted to do and I had already worked it out for the Intel chip. I knew what I wanted to do, I knew why I wanted to do it—it wasn't just what I wanted to do. What I wanted to do didn't matter; it had to be done. It had its own inner logic and I was just on the train for a ride for, maybe, two switches. And the rest was foreordained. I tried to articulate how these things should go and why. It takes a certain amount of time to get the articulation right, and Jerome was picking up on it very, very quickly because he was bright, he was quick. He picked up on immediately and he could understand my arguments and wherever he couldn't he could frame a question which would elicit an answer from me, and so we became partners. This things was as much his as mine. Like any student, I figured once you understand something it belongs to you. If it belongs to you too, that's okay; it still belongs to you.

HAIGH: I think you had implied earlier that the idea of the NaN was something that hadn't been found anywhere previously--

KAHAN: No, the indefinite existed; that was Seymour Cray's idea. Okay, that was on the CDC 6600 and on its successors, the 7600 and so on. You could get an infinity if you divided by zero or if you overflowed. But then if you tried to do something with the infinity, you usually ended up with an indefinite. The indefinite was somewhat different digit pattern. The trouble was that, once you had an indefinite, the hardware was predisposed to trap to what was called an *exchange package*, which was simply a nice way of saying you were gone. However, you could mask off the trap. But if you did mask off the trap then, unfortunately, there is no way to predict what the indefinite would do as an arithmetic operand, because the compiler had not he slightest idea that an indefinite was indefinite; it just did whatever it was going to do, which would vary from one circumstance to another.

On the other hand, the DEC VAX had what was called a *reserved operand*, perhaps inspired by the same thing. Once again, the reserved operand was designed so that, if you did something naughty (for example, overflow), you'd get the reserved operand as a result, if you didn't trap on the overflow. But then if you touched the reserved operand, you were expected to trap. Well, you could put a trap handler in there that didn't do anything, just simply kept on going, but then you could never predict what the reserved operand was going to do next. It would depend on the trap handler. It wasn't really a trap, it was a fault on the reserve operand as you zipped away before you actually did anything with it. So now you have to trap handler to figure out what result to deliver, and then to jump to the next instruction. It would depend on the trap handler; it wasn't defined by the arithmetic and, in consequence, the compiler didn't know what to do with it. So this was the trouble that the indefinite, the reserved operand—these were things that were not so much undefined as inscrutable: you just never knew what the hell that were going to do.

But what I knew was that I had to have an algebraically completed system, because you can't stop a computer. Or at least if you do stop a computer you are sure going to regret it sooner or later. So we had to figure out—I had to figure out—what do you do when you divide infinity by infinity? What do you do when subtract infinity from infinity? And of course that had to depend on context. So the idea was to algebraically complete the system you create a thing called NaN, whose meaning depends on context just the way everything else in the world has a meaning that depends on context: what does *three* mean? We've already discussed that: its meaning depends on context. NaNs meaning depends on context but now you're got a rule that says when you're going to get a NaN, and you've got a rule that tells you what's going to happen if you try and use the NaN. Of course, not every function is defined for a NaN.

But all the other arithmetic operations are; all the algebraic operations are; they are defined by the rules that I laid down. These rules are designed with malice of forethought so that they will both be mathematically coherent insofar as they can be. And they'll be more or less compatible with languages such as they were at the time where the hardware may have a NaN, but the language doesn't. In fact, on the CDC machines, the hardware had an indefinite but the language didn't. On the VAX, the hardware had a reserved operand, but the language didn't. Of course not, it was Fortran. Fortran didn't have any word for an indefinite or a reserved operand. So I had to say we would have to predict what the NaN does in such a way that you can detect it. Everything was hemmed in. All I was doing was discovering what had to be done.

HAIGH: I will ask you—but I think we should talk about some other things first—about how you feel about the kinds of support that compiler writers in the end did add for these things.

KAHAN: Yes, Apple jumped onto that really quickly.

HAIGH: I guess there are a couple of other things we can talk about about the committee dynamics there. One of them is this personal question. It's clear from what you've said before that you don't see yourself as being a natural committee person, so what was it that you saw at this second meeting of the group, the first one that you had attended, that made you think this was the rare committee that you were prepared to invest yourself in?

KAHAN: If I don't do it, it will be done badly. Call that *chutzpah* if you will. But I knew from past experience what committees did. Remember the Chancellor's Advisory Committee and others? I knew what committees did; I realized that what this committee decided would be on our backs in the foreseeable future. Not just my back, but on the back of my children, my grandchildren. So I was stuck. I could despise committee work as much as I liked but, the fact was, this was one that I had to work on.

I am grateful that I had Coonen to help, because I would have been altogether too abrasive to get along in a committee. But Coonen articulated ideas not only well, at least as well as I could in most cases and often better. But he did it in a way that did not abrade and so it was also prudent for me to let Coonen speak. If he was going to discuss a technical question that he could answer just as well as I could, if not better, let him do it. And he did, and I'm grateful for it.

HAIGH: So you had to practice being quiet in certain--?

KAHAN: *Biting my tongue* is the phrase you are looking for!

HAIGH: So Pete Stewart also shows up in the story at this crucial juncture of writing an evaluation to DEC that decided that, actually, your scheme was quite a good one.

KAHAN: Well, on balance, he figured it was the better way to go.

HAIGH: So had you had contact with Pete Stewart prior to that?

KAHAN: Oh yes, but not about this topic. Yes, we had met at various meetings. He was a student of Householder. He was pretty good. He had a tendency to long, drawn-out algebraic work but, later, he got interested in history of various things; as he practiced exposition it got better with the passage of time. And he was obviously thoroughly competent. When DEC selected him to review the contending proposals and express an opinion, I think they had selected an appropriate person. He had experience with software and software packages and development. He had experience with error analysis. I don't know whether if he knew as much about circuit diagrams and exactly how the logic works as I did, but it didn't matter. He was a good person to choose for that purpose, and I did not know what he was going to report. I did not attempt to lobby him because I didn't find out soon enough to have lobbied him, if I had been so inclined. I think he gave his report at a meeting in Boston, and I think we knew he was going to give a report before we knew that DEC had commissioned him a report. I certainly wasn't going to call him and lobby him and tell him what I thought ought to be in his report.

HAIGH: So did meetings of the committee move around the country?

KAHAN: Only a very few times…maybe it was just once, when went to Boston.

HAIGH: So that would tend to bias it towards people in the Silicon Valley area?

KAHAN: You are so right. It didn't stop people from coming a very long distance. They did and, of course, it didn't stop me going on trips from time to time to lobby for people's attention and, I hoped, support. So I mentioned there was essentially a meeting of numerical analysts somewhere in Austria, somewhere near Vienna. Almost unanimously, people said, "Kahan, you're crazy and you'll never get the manufacturers to do all that. It's just visionary counsel of perfection." But that meeting spawned a letter from Jim Wilkinson. He did consider the issues carefully. He wrote a considered letter to the committee supporting what we were doing. That carried a lot of weight because people had heard about Wilkinson; even if they weren't error analysts, they had heard about Wilkinson. I think there were, perhaps, a couple of other letters too; I don't remember exactly who wrote letters. On the whole, the letters favored the KCS proposal. Again, I believe it's because there was something about the coherence of the proposal that made sense to people. They were so browned off with Stan Brown's model. It was adopted in ADA, but people discovered they were struggling with it. They couldn't prove programs worked, even though it was perfectly obvious when you ran them that they were working; the model wouldn't support sufficiently tight characterizations.

HAIGH: So by the time that the standard was heading for approval, were you or other people on the committee hoping that it would be applied to minicomputers or to larger systems, as well as to microprocessors?

KAHAN: It wasn't a hope; it was a confident expectation. Already from my visits to IBM as early as 1967, there had been the predictions that IBM would be in serious trouble when the computer comes in a little business envelope. So we knew it was going to happen. We knew the process: you can't make things fast unless you make them close together, and you can't make them close together unless you make them small. There is no point in having a mainframe or a super computer unless something in it is pretty fast. So the processor was going to be a microprocessor, no matter what the rest of it was.

HAIGH: So it wasn't that you hoped that Cray would redesign their machines at some point; it was that you thought eventually that would just have to make machines out of microprocessors.

KAHAN: Yes, and they did.

HAIGH: So by standardizing the microprocessor, you just had to sit there and wait for the microprocessors to--?

KAHAN: It was a voluntary standard; it wasn't as if we were forcing people to accrete to the standard. Although there were some government agencies whose purchasing departments insisted on conformity to all applicable standards and I had not realized until well after this effort how coercive that was. It certainly acted in our favor.

HAIGH: In terms of implementation of this, I think you've written somewhere that rapid adoption of the standard took place in 1984—before the standard was even formally approved.

KAHAN: Motorola announced theirs in 1984, that was the 68020 coprocessor. The 68881 and 68882 (somewhat faster) came along immediately after. They had already a chip, a 6839, which was the ROM to make the 6809 do this arithmetic. Somewhere I've written down a list of how many people adopted this thing. National Semiconductor had at least two implementations, one somewhat grotty and one rather better. We will come upon that again because they were paying Kwok Choi Ng to write transcendental functional library for one of their little coprocessor chips. You have mentioned Weitek—I've lost count now. Oh, Zilog—I've lost count. These were heavyweight guys who were going to accrete to this standard. Part of the reason I'm sure was because the managers thought it was the standard—what can a standard be? It's something that everybody could do, and they thought it would be easy. They would tell their engineers to build a standard. The engineers would come back later and say it was getting hard. But that couldn't say that it was too hard, because that might imply that they weren't competent. So they had to be circumspect about saying that it was hard and that was one of the motivations for coming to the meetings. They could come and ask their questions and they would find out.

That was why Cleve Moler came up with this famous remark. He said that whenever they had visitors from Europe and they had time to sightsee, there were two things they absolutely had to see. One was the Grand Canyon and the other was a meeting of the IEEE P754 Committee.

[Tape 10, Side B]

HAIGH: Do you feel that the implementations produced were generally of a high quality on the hardware level and delivered the benefits--?

KAHAN: Well, the benefits that were delivered were primarily that you had a predictable arithmetic for doing the kinds of things that you previously were able to do with a reasonably predictable arithmetic—except now, the prediction was somewhat more reasonable for how things rounded; you knew where the bits were and how big the numbers were and the ranges and so on. You didn't have all these imponderables and you didn't have these bizarre end cases. But gradually we found that the compiler writers were misunderstanding the standard. For example, the early Sun machines, based on the 68000 architecture, their compilers were using the extended precision of the 68040 kind of architecture—it was actually 68081, initially. They were using the extra-precise registers to perform all their arithmetic, but it was a registered machine; whatever they spilled, instead of spilling in the width of the register, they spilled to an error in the double-precision format and would subsequently re-load it to continue.

So now what we had was schizophrenia, because you had a certain variable or sub-expression which had one value and one context but, then, in another context where they reloaded it, it had a different value. Needless to say that does not auger well for your algebraic consistency. The programming language community whom I had expected would figure out what to do, in fact never did. I guess that was a lapse on my part. I really should have taken more care to figure out what kind of guys I was dealing with in the programming language community, and I should have acted sooner to provide guidelines with some kind of force of law, not merely suggestions of what should be done.

I think I've got a copy of a paper here by Charles Farmer as he tried to write these things out. But that was just a paper published and which disappeared into the fog of published papers. It would have taken more than that—I would have had to go and work with the language standardizing committees and lobby there, and I had no credibility there. The languages were now in the hands of language architects, and I've told you already what an architect does. Their criteria for languages were mostly aesthetic, although there were people who tried to be analytic about programming languages. The fact is people have these favorite notions about what a language should be like or what a language should do, and they let their favorite notions dominate their decision process. Instead of deducing what had to be done, they already knew what they wanted to do and tried to somehow fit everything into it. It's very, very difficult to have discussions with these people, even if you succeeded in persuading them on Tuesday that you really should do it this way on the left instead of that way on the right, by Thursday they were back to doing it that way on the right. So it was an overwhelmingly difficult challenge and I didn't know how to deal with it. And I didn't have the analog of Coonen with credibility in the language world to work with.

HAIGH: So then in terms of achieving benefits of giving programmers more feedback on what was happening and allowing them to specify how they wanted things to be dealt with, you feel that the power that was built into the standard has for the most part not been exploited?

KAHAN: Yes, and I'm writing that repeatedly. I think the single person who did the most damage in one moment was Bill Gates, Jr. I went with a group of Intel people to Redmond, Washington; I believe it was 1982, but I don't remember the exact month. We came there with the expectation that we would be able to help the compiler people in Microsoft figure out how best to support the 8087. Of course this was before the standard had been adopted, but it was in my view, anyway, with a view to getting them to support the capabilities of the standard, which the chip pretty much did. Across from us we had a group of people who looked interested in the idea. I think that, at that point, we were convinced that at each side there were folks who were well intentioned and competent. So we were going to get along.

Until about 11:00, and in comes Bill Gates and he says, "There is a socket in the IBM PC, and that is where the 8087 should plug in." But he said, as if he already knew it all, "Almost none of those sockets are going to be occupied, and so it is not worth our while to go far out of way to support the 8087. Rather, we should support the software arithmetic very much like we are doing, except possibly modify it to the format that's on the 8087 instead using the format we have been using." And then having made this pronouncement, he walked out and a cloud of gloom descended over all of us, because if that was what he was going to do, then we knew what the Microsoft folks would have to do, and it would kill the 8087. Do you want me to go into the details of what they were doing?

HAIGH: Let me ask you a big-picture question first. Would it have been possible to have Microsoft to have written libraries that would emulate the 8087 if it wasn't there, or just pass the functions through to the hardware if it was there so that you would have code available both ways?

KAHAN: Well, yes, okay. Now you've asked the question that is going to give this five- to ten-minute explanation—sorry about that. The way Microsoft did its arithmetic was to have floating-point pseudo-registers; actually, it was a floating-point pseudo-stack—that means an array in hardware treated as a stack using the stack management instructions of the 8088 and the 8086. If you wanted to evaluate an arithmetic expression, you would push your operands, which means copy them, from wherever they were in memory to this stack. Then when you had enough operands on there, you would then execute the instruction that you wanted but calling the appropriate subprogram that would then replace the top of the stack by a result, possibly POP things. And that was an easy way to manage this sort of expression handling. The subprogram that you called would be one that was designed to simulate floating-point arithmetic, but on a format initially not IEEE standard type or Intel. It was just some format that the guys at Microsoft had chosen. It looked a lot like a PDP-11 format.

They supported only float and double, that is four-byte and eight-byte wide floating-point arithmetic. That is all that they support for. This pervaded their whole compiler all the way through. Everything was like that—just these two formats. With the 8087 there, what they could do was change the format in which they encoded their floating-point numbers so it would look like the 8087s, and change the subroutine that you called to act on the things on the stack, so it would operate on that format instead. That would be easy. Then for all software floating-point without an 8087 plugged in, in effect, they made almost no change to anything else—they were the only changes really to that subprogram. People would have to recompile and they couldn't use their old data, so there was a mode switch: do you want the old way or the new way? Other than that there is no change to the compiler.

How do they do it when you plug in an 8087? In that case, when they call the subprogram to do the arithmetic, it's a different subprogram. Instead of doing the emulation of floating-point arithmetic, what it does is copy the operands from the top of their software stack to the 8087 stack. Then performs the arithmetic operation, pops the stack and stores back into their stacks. Okay, so you see all this movement? On an 8088 with a one-byte wide bus, moving four bytes and eight bytes was a pretty slow business. Moving ten bytes, they decreed, was prohibited—so they were not going to support the ten-byte format at all. Despite the fact that, at least initially, when they did the operation on this 8087 stack, the initial result was a ten-byte wide result, which they would then round again on the store operation. So here was the 8087, crippled because there was all this movement between registers. On the 8088, it was hardly worth having an 8087 if you're doing all this memory movement. So you couldn't get access to the interesting things in the 8087 because they weren't in the original language. All you got was the fact that, now, the hardware was doing some of the work instead of software. That's all, and that's what I mean by killing it.

Now Borland took a different tack. Roger Schlafly…the same sort of name as Schlafly integrals. In fact, his mother, Phyllis Schlafly, is a well-known right-wing political activist (but Roger is a little bit embarrassed by that; his political views are not exactly consonant with his mother's). Roger Schlafly, a Berkeley graduate, was working for Borland; what they were doing was developing a spreadsheet called Quattro, which turned into a very good spreadsheet even though it had a number of difficulties because it became embroiled ultimately in a lawsuit with Lotus 1-2-3. Ultimately, Borland won that lawsuit but it was a pyrrhic victory. I don't have to go into that right now, but Roger said to his friends who were developing the C compiler at Borland; which compiler he was going to use in order to help program up the Quatro spreadsheet program. He said, "I'd like you to support the stuff on this chip because, if you do, I can make the spreadsheet better."

So Borland's compilers came out with the ability to declare a four byte, eight byte, and ten byte floating-point numbers. They were float and double and long double. They provided a certain amount of support for the elementary transcendental functional library by copying, insofar as they were allowed to copy at some of the Intel code that supported the library. I guess they must have licensed it. And they were reasonably conscientious.

They weren't the first, because Apple was the first. Apple was actually doing it on the Apple III, entirely in software. You'll remember UCSD Pascal on the Apple III. And then of course when the Macintosh came out they were doing it entirely in software and, ultimately, they got these lovely little coprocessors, and then they were exercising them and they got them to go faster. They had started them before Roger Schlafly did at Borland. The Quatro was supposed to run on an IBM PC. It was supposed to run on them whether it had a coprocessor or not in the socket. That mean that the *C* that you got from Borland had an 8087 emulator, which actually emulated with reasonable fidelity. They didn't support everything in a convenient way, but they did provide access to the control word and the status bits in the 8087 so that if you really wanted to (and you'd have to want to pretty badly), you could get these extra capabilities—the directive roundings and stuff like that. That produced a pretty good spreadsheet. I think I still have Quatro and, indeed, occasionally I still use it. I think it's preferably to Microsoft's Excel because it doesn't have all the bugs in it.

But gradually, over a period of time after Roger left Borland to live in SoCal now just south of Santa Cruz, gradually the folks at Borland, who did not understand what Roger did and why he did it, would put in various kludges, and I've written about those in, I believe it is, *Marketing vs. Mathematics*. I give an example of how Borland just misinterpreted what they should have done in response to some complaints. Well, for a little while, Microsoft felt that they had to respond to the challenge from Borland but, I believe that because of the lawsuit, Borland fell behind in the compiler market enough that Microsoft no longer felt that it was a significant competitive threat to their C compiler. I think Borland's C and C++ compilers still support the extra-wide format, certainly did last time I looked. I have one of those compilers around, not the newest.

Borland started to concentrate on programming development environments, which they could produce in a way that was more humane then Microsoft did. So they did not continue to develop their compiler and, consequently, it's not the fastest compiler in the West. Microsoft dropped support. It was grudging to begin with, and they dropped that support when they felt Borland was no longer a significant competitive threat. In any event, when they went to Windows NT as they knew they were going to, they were going to be running on the Alpha, and the Alpha didn't have three IEEE floating-point formats. The Alpha had five floating-point formats. Two of them were IEEE—that is, floats and doubles—and then on the Alpha, if you wanted, you could support the DEC formats F, D, and G. And in software H, which would be a SIC format supported somewhat reluctantly in software.

So how many does that make? Six floating-point formats. But Microsoft is going to support only two of them, and did. That meant that they killed the vestigial and reluctant support for the 10-byte format. And they turned off, in their subsequent compilers, Windows NT and-- in fact, after Windows NT appeared, all their compilers, so far as I can recall, killed off all support for this extended format: even to the point of setting the rounding control bits so that you couldn't get a 10-byte result, insofar as its precision was concerned, no matter what you did. It would then simulate in Intel hardware the floating point that you could get on the DEC Alpha hardware.

HAIGH: So I think I may see an irony here. Let me state it and then you can react.

KAHAN: Just one irony?

HAIGH: The irony would appear to be that, with this standard, you were trying to do two things: one of which was to standardize the arithmetics so that software would be portable. And the other one was to provide programmers with access to wonderful numerical nuances that would give them more control and accuracy. Yet, with this compiler situation, it seems that the price of a programmer exercising this control would mean that their code wouldn't be portable to other compilers and, presumably, to other machines because there was no standard way for compilers to expose this functionality.

KAHAN: That's absolutely right. The standard was intended to standardize the programming environment, not the hardware. And we failed because, I think now, wrongly: we did not think that we had authority to pontificate on programming language matters. But we should have, and we will. We are not going to make that mistake again. It may be that, in consequence, we are going to get nowhere because the programming language community may ignore us. That may happen. But we are going to try. If we fail it will not be for lack of trying, as was the case for the first issue of the standard in 1985.

HAIGH: So we will return to that topic then. I just wanted to ask you about the lesser-known 854 standard, for different word lengths and radix.

KAHAN: Yes, I was going to pull out my HP-71B and put it on that list there. You came just before I yanked it out. I can pull it out if you'd like to take a picture of it. The guys at HP with whom I was consulting thought that now that they have this 854 standard, why don't they produce a device that conforms to it. It was nice to have; I know that there are several people who have these things and like to use the capabilities that were built into the standards—the directive roundings and all these other things. But it only accepts Basic, or a kind of assembly language; it runs slowly; it's got a very limited window. It's got sort of fraction-of-a-line window, so you can't even get a whole line of Basic in there sometimes. So it's not a pleasant thing to use unless you have relatively simple computations.

If you get the MathPack it will do some nice things. It will do matrix computations; I don't think they will do them as nicely as the HP-85 did. I couldn't get through to the matrix guys, so we had gotten through to my friend, Homer, on the HP-85, where he put in the right things. The guys doing this MathPack for the HP-71B were doing it off somewhere. I didn't even know where they were or what they were doing, other than that they took some codes that had already been used on HP-85, since their arithmetics were very, very similar. They took some programs from the HP-85 and redrafted them for this 71B. The 71B had the ability to interface to an instrument bus, which became standardized by the IEEE. That meant it control laboratory instruments and, in fact, that's where it flourished.

But also, it was expensive. You could afford it to control laboratory instruments—but if all you wanted was a handheld calculator, that was a lot of money to pay and it was easier to use a PC. You have to remember when it came out, which was 1983 or 1984--

HAIGH: You have a date here of 1983 for the machine, and 1987 for the standard itself. So I guess it was before the standard was official.

KAHAN: There was always this delay in getting the standard official; I mean the drafts for 854 were done pretty much in 1984. After all it was a publication that came out not long after that. The publication of the proposal certainly came out before the approval.

HAIGH: So was that the only machine that was built to the standard, or did other people take it up, too?

KAHAN: That's the only one I know about whose hardware was designed for the standard, and even that is a little bit stretching because, of course, this is actually a very rudimentary microprocessor microcoded from ROM to perform arithmetic in a certain way.

HAIGH: So what was your personal involvement with 854? Were you one of the main forces in pushing it, or was it something other people were doing?

KAHAN: Well, in 854 I had a certain amount of experience that I couldn't use too directly from this work on FS at IBM. But there were some obvious things about parameters, and 854 sort of wrote itself. It just took 754 and substituted decimal, and allowed for some parameters. There were some formulas that had to be put in place of explicit numbers.

HAIGH: How would you describe Cody's work on the project?

KAHAN: Well, Cody was the chairman and, I don't know, I think he was a little bit surprised to find himself chairman. But he tried to run a tight ship, and he understood the issues and stuck pretty closely to 754 with parameters. That is instead of radix 2 it's radix 10. Instead of so many significant bits, it's number of significant digits, which you chose. We had to do it that way because, at the time, nobody had settled upon a word size for decimal arithmetic. Decimal arithmetics were being performed always in software, but they were being performed with varying word sizes and, mostly, it wasn't floating-point anyway. If it was floating-point it had varying precisions—different in this place than in that. So you couldn't really say that you needed to standardize on the word size.

In any event, what is perhaps more important, decimal did not need a standardized interchange format so badly as binary did. In decimal, you could exchange your numbers in a directly viewable or directly printable format. It would be probably ASCII, or it could be Unicode. If you really wanted to pack things tightly, you could pack them in BCD. If you packed them in BCD, it's going to be harder to display them without a modicum of programming translation. But it was so easy to do that program translation; you just about do it on the fly and it's not a big thing. Best of all, it was WYSIWYG—what you see is what you get. So the need for standard word sizes was not so intense, and I'm not sure it's intense now, either, I don't know. I think people who think they need it are probably mistaken. So: less stuff to do.

HAIGH: So at the time, as everyone went to the bother of producing this standard, was it just for neatness or did you have an expectation that it might be more widely implemented?

KAHAN: Well, I really wanted to look at it as a guide: what you should do if you would like to implement decimal arithmetic in such fashion as would permit an exchange of software, possibly after recompilation. As far as an exchange of data was concerned, I did not expect that the format had to be specified by us, since there were only a few formats worth using, and there were conversion programs among the formats. I figured that we might as well leave that alone.

Whereas, in the binary world of scientific and engineering computation, people had enormous amounts of data in formats they could no longer read because it would be on some tape—and they don't have the tape drives anymore. They don't have the software that knows how the bits are on the tape. It was a real pain in the ass. What are you going to do with that data? You still have maybe one computer that can read it: to *what* should that format be converted? Well, if you have the IEEE standard format, then that was something that looked as if it would last for a while, so the binary folks needed it. Furthermore, the binary format had the happy property that, in many situations, its precision and its range majorized what was already out there. So that meant you really could copy your binary stuff and get exactly the same numbers in very many cases. Maybe even with the same word sizes.

But in decimal, there was such a reckless variety of decimal formats that we didn't think we were going to do much good, and I still have doubts that we will do the world much good by telling everybody they've got to pack their decimal numbers into a certain format exactly this size, that size, or some other size. Just give them three sizes and that's it. I'm not sure we're going to be doing him a favor if we do that.

HAIGH: Let me wrap this up then by asking about your ACM Turing award which, I understand, was given for the work on the IEEE standard.

KAHAN: I think so, yes.

HAIGH: So was this a surprise to you?

KAHAN: Oh, it was. I'll say. I certainly didn't solicit it. I had no idea at all that it was coming and in fact at first I wasn't-- I thought maybe somebody was playing a joke or something. I really didn't know what to make of it.

HAIGH: Did you get a phone call from the ACM president, or something like that?

KAHAN: Something like that, I don't remember exactly how it happened. I think it was a phone call first and then a letter. We had to turn up somewhere to pick it up, and I had to wear some stiff clothing. I think, perhaps, some (I don't remember exactly which) family members attended. It was really quite a surprise and, yet, after I thought about it for a while, I came to realize that, in some ways, this was merited except for one thing, and that is that they gave it to me, but it would have been right to give it to a bunch of guys. I said as much. I said I guess it's great to get an award for work that was actually done by my friends and ex-students—because the work *was* done by my friends and ex-students. I was on a train; I was deducing what needed doing. These other guys were doing a great deal of work in order to persuade people, in order to implement it, and so on. So in a certain sense I was just a figurehead and, in a way, I still feel like that. I don't feel that I was Michelangelo. It was something that had to be done and it was easy enough. If anyone else had come to think about what had to be done with my background experience and acquaintance, I would like to think that they should have come to pretty much the same conclusion.

HAIGH: So would you rather have won it for your mathematical work?

KAHAN: What do you mean rather have won it? I would have been content not to win it at all. My mathematical work? Well, I had already received some awards for the mathematical work. Remember I told you about the Forsythe Award? And Demmel and I got some hotshot paper award for the SIAM thing. I think I didn't write very many papers, but the ones I wrote were pretty well…I think Gauss put it this way: "Paucca sed matura" [few but ripe] And I certainly match the paucca part. About the matura part, I'll have to let someone else judge.

HAIGH: Would you say that winning the award had any impact on the way your career developed subsequently?

KAHAN: Well, I saw people deferring to me who previously wouldn't have noticed me. After all, I walked around looking like a janitor. They offered me the possibility of a reserved parking spot, and I said I didn't think that was a good idea, because the reserved parking spots are normally unoccupied. So all I'm really going to be doing is removing one more parking spot. I wish, if you really want to honor me, why don't you build a new parking structure and stick my name on it? I would prefer that, by far!

I think the best comment I can make on the sequence of awards that I have received—because it's not just the Turing award. There was this John von Neuman lecture, which has certain prestige; there is the Piore Award from the IEEE. Actually, that is a very valuable award, because it takes account of the fact that I had placed some weight on social responsibility to try to do things—not just something technical, but something that would have a societal benefit. You have heard me say that. I really don't want succeeding generations to have to relive our experience, and I think they may have influenced the people who decided on the Piore Award.

One of my friends told me a Russian joke. It's about a diplomatic reception; there were mayors there and ambassadors and military figures wearing medals. Of course there were also some pretty, young girls, who were brought along in order to make it more interesting, rather than have just a bunch of old men and, perhaps, some old women. Among the guests was a very much bemedaled Russian general, and he was approached by one of these young beauties who said, "Oh my, General! You have all those medals. You must have been an extremely brave fellow." And the general said, "Not exactly. This medal here was a mistake. Then the rest followed in logical sequence." And I think that's an all-too-apt description of how things work.

I asked my colleagues not to put me up for awards. For one thing, I feel that I have a few and that's enough. For another, I think awards are needed more by the young people than the old ones. But they don't listen to me, so I was put up by my friends in the math department, despite my protests, for the American Academy of Arts and Science. It's not the American Association for the Advancement of Science. Now that is one which I have a great deal of respect for. But the American Academy of Arts and Science writes various things that are supposed to be learned and, every now and then, some of them are actually quite interesting—even some that are altogether out of my field, which, nonetheless, I read.

And then there is this thing for the National Academy of Engineering… I don't know how that happened. I don't know whodunit; I hope it wasn't my colleagues in computer science and engineering, because I asked them not to do it. I don't know if I'm going to go out there in October and all this stuff because, if I do, I'm going to have to wear a jacket and a tie and things like that which I really dislike.

But more than that, I'm going to have to leave my classes for a few days and we don't have a hell of a lot of time to do what needs doing in my classes and I really hate traveling away from my classes. I think my students should believe that they are among my highest priorities while the semester is going—rather than they are just something I do and, as soon as I get an opportunity to flit away, why, I'll do it. I really don't want to give that impression to my students. So I don't know what I'm going to do about the National Academy of Engineering. I guess I'm going to be a part of it, whether I attend the ceremony or not. But I have other things to think about now.

So in my feeling, the Forsythe Award was an important one because I really admired George Forsythe, and although I don't think that award has been offered much, if at all, since then, I like to remember him as someone who encouraged me at a time when I needed encouragement. In that sense it's the right award. And I think the Piore Award is, perhaps, an appropriate award because of my feelings that we do this because we have sort of a moral obligation to do it. It's really the right reason for doing it is because it's going to reduce costs. It's going to reduce costs for most people, even if it increases them for a few. I've got some honorary doctorates here and there: a couple at Waterloo and one from Sweden.

So what he did was get me to commit to give a talk first and, of course, to talk about the IEEE standard as we are revising it, absolutely! Once I agreed to do that, I discovered that I was going to be the object of dedication and so on, and I was terribly embarrassed. Once again, I'm not the only one doing this. There is an enormous support structure. This is part of my support structure, too. Without Sheila, I'd be a lazy bum.

There is a certain injustice in a scheme in which we give awards to a person rather than to a person and the group which accreted around him—or, I would prefer to think, accreted around an idea whose time had come. I think that's the way the awards should properly be posed. But the award I got with Jim Demmel for a paper on the SIAM things—that one was fair-and-square. [SIAM Activity Group on Linear Algebra, Prize for 1988-90's Outstanding Paper]. If we hadn't done it, chances are nobody would have. It wouldn't have occurred to anybody, even though it seemed so simple. All we want is to get numerical results that are at least about as good as the data deserved within the limits imposed by the resources available. What more can you say? We weren't getting that until we showed that there were some problems where, indeed, there are algorithms that will do the job right.

HAIGH: Maybe this would be a good point to wrap things up for today, then.

[Tape 11, Side A]

Session 7 begins on the morning of Monday, August 8, 2005.

HAIGH: We've talked through the floating point standards effort and your career as a consultant, particularly with respect to the various Intel math chips. So I think before we delve into other aspects of your career over the time since you arrived in Berkeley, perhaps we should just go back and give some overview of your experiences here. Now you arrived here in 1969…

KAHAN: January of '69

HAIGH: You've talked about Parlett's role, and the state of the department when you arrived, and the fact that pretty soon after you arrived your triple appointment became just a double appointment between the mathematics and the computer science and engineering departments.

KAHAN: No, it was a joint appointment between math and computer science, in the Faculty of Letters and Science. There was a separate Electrical Engineering and Computer Science department within engineering. That was done because the department in Letters and Science was formed partly be breakaways from EE and CS who chafed under the decisions made by the then-chairman, that was Zadeh, who as I mentioned would appoint people according to his own taste and judgment, despite the critical, or from his viewpoint hypercritical, views of some of the faculty. And so the breakaway group formed this little L&S department, and no sooner had I arrived than campus unrest became an important factor in decisions. Campus unrest led to the formation of an ethnic studies department, over the dead bodies, figuratively speaking, of some academics who thought it couldn't possibly have any academic merit. I think that the history of the ethnic studies department has been a mixed story. But it impacted us because the promises made by a dean to provide a certain number of positions, those promises had to be withdrawn because the state budget was comparatively inflexible, but the demands for positions from the ethnic studies department had to come out of somebody's hide, and so we encountered some difficulties in getting permission to recruit the people we could want to recruit.

There was also the question of facilities. At this point the L&S department was housed in what were called temporary buildings; they had been there temporarily since World War II. So by 1970 their temporary quality had been perhaps overlooked. Finally they were turned down, but not until Evans Hall had been built, in fact it was rather after Evans Hall had been built that they were torn down, if I remember correctly. And the L&S computer science department got a floor in that building. The building had originally been intended to house only mathematics and statistics and the library, but soon they found themselves sharing the building with the computer center, and the computer science department, and even now they've got all kinds of things in there because of the shortage of office space and rooms due to seismic refitting and so on. Anyway, as far as our little department was concerned, we were teaching computer science to letters in science students and, ultimately, to others. Our graduate program was for people who, for some reason, did not want to have an engineering degree. They wanted to have a science degree, and for the life of me I don't understand why that should matter. But I have always been insensitive to things. Here is a guy who has been a mathematician for almost all of his life and doesn't really see why we should discriminate against applied mathematicians. I guess I must be blind or something, I don't know.

Anyway, the time came when the campus could no longer tolerate two computer science departments, so there was a shotgun wedding. This took place in 1972–73, when I was off at

IBM. Then it became clear that computer science was going to need a number of faculty that just couldn't be housed in the existing facilities and would have to have a building. So they built Soda Hall. There's a paradoxical theorem—I think it's Tarsky and someone else; I have forgotten the other name [Banach-Tarsky] It's a theorem that says that you can partition a solid sphere—a ball in three dimensions—into two mutually disjoint subsets of chunks which can then be reassembled into two solid balls, no holes, each the same side as the original. Which means that, somehow, you have just doubled the volume by breaking these things into sets.

My office turned out to be an illustration of that. My office had been on the fifth floor of the math building out in Soda. When the computer science division, the part of it that was in Evans Hall got moved into Soda, I ended up with two offices, because the math students I worked with would be disinclined to walk across Hearst to Soda Hall. The computer science students would be similarly disinclined, and even more disinclined now because, as it turns out, there isn't a direct path. You had to go through some buildings or go circuitously because of construction going on now. So I ended up with two offices and, miraculously, the material that had previously filled my original office in Evans Hall now filled the two other offices but filled them in such a way that I could no longer find things. You've seen my office at home, so just imagine two more like it.

HAIGH: Would you care to describe it for the listeners of the tape?

KAHAN: Well, let's just say it's piled high with papers, most of which, I'm sure that 999 out of 1000 should by now be thrown out, except I haven't got the time to go through them to find the one in 1000 that I should keep. The one in 1000 I should keep I frequently do find, even if, sometimes, it takes an extraordinarily long time to find it. It took me some time to find these documents. I don't want to throw them out blindly. There are things in there that I know I need from time to time, and every now and then I go through the bog and I actually do find them. But it's an awful lot of paper; a couple thousand volumes of texts and every day's exigencies require time that I can't spend sorting through the stuff. This committee work has consumed the time I would have spent during my past sabbatical year sorting through the stuff. Sheila has already told you had disappointed she is that I didn't get rid of all those papers.

HAIGH: Did the merger of the two departments go smoothly, or was their some friction?

KAHAN: Yes. The animosities that had caused the schism in the first place had somewhat abated. For one thing, the chairman that we have had for the past several years, let's say they have been elected by consensus and have recognized an obligation to balance the needs of the electrical engineering people who are predominantly in Cory Hall, almost all of whom have heavy interests in one aspect or another of computing, either contributing to it or using it or networking it or whatever. And then there were the computer scientists, many of whom have work that overlaps with the guys in Cory Hall. I think it's fair to say that there is an enormous amount of collaboration and cross-fertilization that goes on. This shotgun marriage turned out to be, I think, on the whole a good idea that has made for a rather healthier department.

The various little interest groups that form and then reform and so on mean that students can generally find a place to work which may involve faculty, not merely from computer science and electrical engineering but also from other places, and get their work done with collaborations that suit the work. The administrative barriers to graduate work are pretty miniscule. What it amounts to is, if a graduate student does pass the preliminary filtering exams in any graduate department, he can then work with anyone in any other graduate department, take graduate courses anywhere he wants, and have practically anyone he wants on his thesis committee, provided he has at least one member from his own department. So I think it's fair to say that that merger didn't really matter.

What really mattered was that people were collaborating. The collaborations crossed departmental lines. For example, Bernd Sturmfels in math has interests at the highest levels of abstraction in mathematics, but he also uses computers. He uses them not only in semi-automated algebraic systems, but he uses computers for very specific computations. He has a sort of zero FTE appointment in computer science, which in effect, gives him the right to join in the computer science lunches. We have joint appointments in statistics and, of course, there's the bioengineering effort and the links of the biology people. Richard Karp, who retired and then went off to the University of Washington in Seattle, finally has come back on the faculty. He is un-retired and very interested in the genome project and so on.

We can't always keep the people…we had Gene Myers for a while, but he is leaving. It's not always easy to keep people, but that is partly because we are crowded. We are really very short of space. There is this new building coming up, the Citrus Building, which will help. But my fear is that it's going to be filled to overflowing as soon as it's opened. Space is more the problem than anything else among the departments with which I am most closely concerned.

HAIGH: Did you yourself ever consider moving on again?

KAHAN: I would get these offers from time to time—feelers—how would you like to go somewhere else? I told you when I discussed moving on, regardless of where we would move on, my family outvoted me. We can't easily leave now when we have four granddaughters on the West Coast. But this has been a reasonably hospitable place to work and, indeed, the consulting activities, which have tapered off quite a bit—I don't feel I need to do them. I don't solicit or cultivate consulting the way I might have in the past. But those consulting activities prove crucial, as you've seen; I don't think that would have happened had I been elsewhere than within a stone's throw of Silicon Valley.

HAIGH: When you discussed your arrival at Berkeley earlier in the interview, you had made a comment that you were very happy because, in those days, you had no idea how marginal computation and numerical analysis were going to become within computer science. That you--

KAHAN: Well, when I came here I had an idea. It was happening, physically happening. What I said was that when I started in computing, scientific and engineering computing were practically everything that justified the big computers. Most of the business administrative stuff was run on the smaller computers, with the exceptions of some very large companies that had some very large computers. I think you could call the oil companies and their seismic stuff—I don't know whether you want to call it "business commercial" or "engineering," but we regarded that as more scientific than commercial and administrative.

But very rapidly—and, looking back on it, it should have been predictably—the business and administrative computing overtook the scientific and engineering computing by some order of magnitude. Even if you include all the defense department-related stuff. That made computers a commodity, which the scientific and engineering community could afford. So there was a pronounced movement away from the batch computing and time-sharing type of computing, towards each little scientific group having its own little computing center around their own computer.

Of course that's extremely inefficient, and it also meant that all kinds of bitter lessons had to be relearned by people who didn't connect with the major computing centers that had existed. I don't know whether now whether Grid computing and so on is going to lead to a re-centralization; I rather doubt it. I except that there will always be a tug-of-war between having your own computing capabilities and in your data on the premises, versus leasing computing resources elsewhere, which may be cheaper, perhaps, in the long run or in the large, but it isn't yours. All these security gaps I'm sure are going to frighten people into thinking that they're a lot safer having their stuff onsite, which probably isn't actually true.

HAIGH: I was thinking also of the place of numerical analysis and scientific computing within computer sciences and academic discipline that, as time went by, it seems that--

KAHAN: There are computer science departments, which have no numerical analysis worth a damn going on within them. It's quite easy for someone to obtain a bachelor's degree in computer science and go on to doctorates in computer science without having had more than one hour of exposure to numerical computation. And that one hour would have been in some programming class where the instructor apologetically had to tell them about floating-point numbers, and that these things are all extremely mysterious—sort of a black art. That is quite a change. In our computer science department, there is an obligatory requirement to have one semester's worth of numerical analysis. I must tell you that it doesn't do a hell of a lot of good. Part of the reason for that is that the numerical analysis is frequently taught by someone from a math department who thinks of numerical analysis in terms of algebraic rearrangements of formulas.

We do have courses of numerical analysis offered not only by the math department, but often by other departments that teach the sort of numerical computations that matters most to them. In many cases what they're really teaching is how to use packages. We use a package, MATLAB, very extensively because it's a good way of getting the job done without having to learn about the peculiarities of programming languages designed for some other purpose. Nonetheless, many engineers find it necessary to learn the program in Fortran and/or C, simply because C is the connecting glue and Fortran is the legacy language. But I've characterized numerical analysis, I think, accurately as a sliver under the fingernail of computer science.

HAIGH: I know there were several other projects that you were involved in. One of them I found a reference to was the production of the mathematical library for the transcendental functions as part I think of the broader BSDF that had been going on.

KAHAN: Well, that's not the way it started. The way it started was that Parlett had a student, Kwok Choi Ng, who had come from Hong Kong on a student visa. He had written a thesis under Parlett on some aspect of matrix computation. He didn't want to go back to Hong Kong because his parents were of very modest means and therefore the prospects of his getting a job that would use his talents were practically negligible. Nepotism helps in that part of the world, and it wouldn't have helped him. So he really had to get a job in the U.S. But in order to get in line for a visa and work permit—I think there was a seven-year waiting list, and that wasn't going to be very good. So Parlett and I decided that he had to get some experience in some of the less-than-fully mathematical areas like parallel computing in order to become eligible for a job working for a firm that would get the necessary visas for him.

So he went to work for me and, indeed, there was some stuff done on matrix computation and other things. But the IEEE standard was coming out, and it was pretty clear that it was going to win. Only the people at DEC thought that they could stop the train. National Semiconductor had produced a chip, which implemented the IEEE standard in a minimal sort of way, and they needed a math library. So since they couldn't employ KC, what they did was contribute money for my researches with the understanding that these researches would lead to a math library, and they had first crack at it. They even sent a microprocessor development system with their chip for KC to work on. And he did. He wrote a library of elementary transcendental functions—actually a pretty good library. But by the time we delivered it, the man who had commissioned it had left National Semiconductor, and the person in his place to receive it had no idea what to do with it. Well, the money was already paid to me, and I paid KC at the rate of support for a graduate student, so he didn't go hungry. But what were we going to do with it?

HAIGH: Do you recall what the chip was called, and what year this was?

KAHAN: It was somewhere between 1981 and 1982; somewhere within that range, I don't remember exactly. I could probably go rummage through some floppy diskettes and look at the dates at some files and find out if it's really important.

HAIGH: No, I just want an approximate sense.

KAHAN: Well, it's 1980-something and it was certainly not 1985, so it's got to be the first half of the 1980s. What we had observed was that the math library in UNIX which had come to us from Bell Labs with the rest of UNIX was reasonably good for the '70s, but had a large number of lapses. There were places were it was clear that amateurs had done the job.

HAIGH: Would that be the standard library that's in those math.h files?

KAHAN: Yes, that's right. It comes with the math.h and the header files.

HAIGH: With the C compiler?

KAHAN: Yes, exactly. It was a library that had been done by folks who, I think, are best described as having something else on their minds. So it ran slow in some cases and inaccurately in some cases. The people were working on 4.3 BSD, which they figured they would distribute principally to VAX installations, welcomed the idea that I might put together a band of students, including KC, who would update the library. Later it became much more important that Berkeley UNIX be liberated from AT&T's copyrights. And then they were really very happy to see that our programs were practically independent of Bell Labs, except for the names and calling sequences, so that people who already had programs going would not have to change their programs. All that would happen would be that their math functions would get better and faster, perhaps, too. That was the only thing that was retained. I think maybe errno had to be retained for compatibility, which just means that certain conditions you had to set a certain error flag. But that was it and the rest was done quite independently.

The reason that it was done independently is because we had programs based on a paper I had actually written at Toronto. That's the one I can't find which is called Superlinear Convergence of a Remez Algorithm. Remez is the name of a Russian mathematician who first put forth algorithms of this kind. The basic idea behind these algorithms is that you want to approximate a target function, like a logarithm or exponential, by some expression easier to compute, you want to choose the parameters in that expression. They may be the coefficients of a polynomial, but my paper was more general than that. You want to choose the parameters in the expression to minimize the maximum deviation of the expression from the target function. That minimized maximum deviation will occur when there are a number of extrema—typically, alternating extrema—so that the error curve is an oscillating curve whose amplitude at the peaks is constant. It's that you've minimized them all; you've compressed them all.

If all you wanted to do was polynomial approximation in the usual way, it's easy to prove that such an approximation exists and that you can find it in a certain way and so on. However, that characterization leads to the construction of an algorithm—not altogether an obvious algorithm, I might add—which usually works in a case that John Rice characterizes in his book as the unisolvent case. But the unisolvent case is honored more in the breach than the observance. With a non-linear approximation (Rice mentions that, and he is right about that) it turns out that you can write up a program that usually works. And it is worth employing in an area where you can plot the graph of the error in order to see that you aren't being fooled. You can be fooled: if you want to get the best rational approximation ratio to polynomials, you can go through a process that will equalize the amplitudes of the oscillations, except that in the middle you can have one of these poles. It's a spike that goes off into infinity in one or two directions, and it just so happens that that's not regarded as an extrema, because the graph doesn't sort of turn over, come up to a maximum and turn over, it just goes off and never goes into the algorithm. So you can be badly fooled.

Naturally there are ways of dealing with this; the easiest way to deal with it is to plot the graph of the error and look for spikes and other things. We were coming to an era when plotting looked as if it might be reasonable. Initially it would be printer plotting; I don't want to go into that, it's just too horrible. But ultimately, plotting as we've now come to know it was coming into vogue, and that meant that, looking forward, I could see that a certain kind of algorithm would be worth working on. What was interesting about the algorithm was that, if it worked at all, it worked fast. This means that, if it wasn't going to work, you would know about it soon; if it was going to work, it would have already finished. So this is a desirable situation, because if you can't have something that works all the time, it's nice to know that you'll know soon that it's not going to work. That's next best, let's put it that way. So this algorithm had that characteristic.

I brought that algorithm with me, and the students who worked on the project with me (I think I've listed their names, or at least most of their names, on that document), some of them became custodians of the algorithm. I think Stuart McDonald, in particular, was one of the custodians. By *custodian*, what I mean is that the students would learn to integrate the automated algebra systems like MACSYMA and then, subsequently, you could use Mathematica or Maple. But then we were using MACSYMA. You use a symbolic algebra system to evaluate derivatives symbolically. That turns out to be an alternative to schemes which, given the program for a function, can generate a new program that computes the new function and its derivative simultaneously. If you have one of those, that would be okay, too. With that armory of equipment, you can find the best coefficients, which give you the minimax approximation. Jim Cody has described it; he uses the word *Chebyshev approximation*. But *minimax approximation* is a better word because there is another meaning for *Chebyshev approximation* in terms of what are called *Chebyshev expansions*, and I don't want to go into it.

Still, there is a headache. I told you about those parameters, for instance, choosing optimum values for the parameters. Well, let's take a simple case: the parameters of the coefficients of a polynomial. You've decided a polynomial degree five will do the trick. A polynomial degree five has six coefficients, and you want two six coefficients to minimize the maximum excursion. Unfortunately, your function may have some property which imposes a constraint. For example, it may be that the value of the function when $x = 0$ has to be one. It has to be exactly one because, otherwise, a because otherwise a cosine doesn't work properly if the cosine of zero isn't one. So that nails down one of the coefficients. When you've nailed down one of the coefficients, you run the risk that, now, the general theory doesn't work. But, nonetheless, the Remez algorithm would work most of the time.

Then you've got another headache. This headache is the fact that the functions that we are dealing with are transcendental and, therefore, the ideal values of the coefficients are transcendental numbers, which means that don't fit perfectly into floating-point. You've got to round them off. Well, think of the sources of rounding error that you've got. Aside from the fact that your polynomial rational function or whatever (because we could use whatever), it didn't have to be polynomial rational. Let's say it's a polynomial, and you find that, first, there is the error that you incur because the polynomial is not the same as the target function, and you've minimized the worst of that. Then there is the problem that, when you evaluate the polynomial, the actual arithmetic that you do for any particular argument will generate rounding errors. And then there is the fact that you're evaluating the wrong polynomial, because you can't have the optimal coefficients. You've got to round them off in order to fit in the computer. What are you going to do about all this?

This non-standard Remez algorithm allowed you to take advantage of property of minima. When you minimize a function, sometimes the minimum occurs in a notch, like the bottom of the letter V. But more often, the minimum occurs and is distributed like the bottom of the letter U. If you don't get exactly at the bottom of the U—if it's just a little bit to one side or another, so it isn't exactly the minimum, but it isn't much bigger. What that meant was that we could round off one of the coefficients whose rounding error would have damaged the function worst and then adjust the other coefficients to compensate. The Remez algorithm talked about still worked.

Of course you have to plot the graph because now, instead of having oscillations all of equal amplitude, perhaps, one was lower than the others because you didn't have enough degrees of freedom to compress the others and let that one come up. So the others would be a little higher. You have to plot the graph in order to see that that was going on, but still the main thing about the algorithm was that if it was going to work it was going to give you an answer and do it fast. And so you would know really quickly whether you could get this.

HAIGH: Presumably if it wasn't working, you would fall back to a slower method?

KAHAN: No, if it wasn't working, what you really would do was decide that maybe you should fix some other coefficient.

HAIGH: So was the effect of this that the library you produced was unusually fast and retained accuracy?

KAHAN: It was unusually accurate, because half at least of the error in the final product occurred because you couldn't have the right coefficients. We could get the coefficients; they weren't the right coefficients, but they were coefficients that had fit in the machine and been adjusted. And usually this would worsen the discrepancy between the polynomial we compute ideally and the target function—worsen the discrepancy there by a very tiny amount. So, in effect, we'd gotten rid of half—almost—of the contribution due to round off. That meant that our functions were unusually accurate for the kind of function.

But we did more: we chose expressions, which had the property that we would suppress most of the rounding errors that would occur during the evaluation. It was the kinds of formulas we chose; they weren't necessarily the obvious polynomials. So we chose these formulas and we ended up with a library which, for most of its range, had errors significantly less than one unit in the last place. This is rather better than can be achieved with a Codyian weight. But Codyian weight didn't have all of our tools, and they didn't have the arithmetic that we targeted.

We targeted DEC VAX arithmetic or IEEE 754 arithmetic. No other arithmetics need apply. We weren't going to do it for hexadecimal; we weren't going to do it for Cray's; we aren't going to do it for Burroughs' machines, and so on. Only DEC VAX arithmetic and IEEE 754. The programs for these two arithmetics were so similar that…these use the same numerical coefficients and stuff. It's just that we had to shift the exponents a little bit because the encodings were slightly different.

The exception handling was very slightly different. So we had these two libraries. They were fast and usually accurate and in the public domain. You could get them from 4.3BSD UNIX by just asking for the tape. If you wanted our library you would just ask for the tape and you didn't even have to get the license from AT&T. And they were incredibly widely copied. I know that that had quite an impact, but the impact was limited to some degree because, at the time, memory was still expensive. I had hoped that our functions would be or could be incorporated into coprocessors and that they could be put into ROM. That hope was, perhaps, misplaced. We didn't realize how rapidly memory prices were going to decrease. And we didn't realize how unreasonable it would turn out to be to have coprocessors for floating-point when you have zillions of transistors available and you can put the floating-point on the same chip. You can't put the ROM on the same chip because the space occupied by the ROM is better occupied by cache. So that meant, in effect, that large tables could be used and that would incur different algorithms.

That was what Peter Tang went into to. By the time Peter Tang was rewriting these functions, he was able to use large tables and he didn't use the same techniques. He didn't have to. But the programs we developed might come back if the day comes that folks say no, you can't have big tables, because in order to get to the big tables you have to go off the processor chip in order to go to memory. The energy involved in getting off and back and so on is more than our battery can stand, et cetera. So who knows, these things may come back. But in the meantime, there was a library that set a standard. In effect it said, "If your functions are worse than these, why aren't you using these?"

HAIGH: When you say that they were very widely copied, do you mean that they were widely used for UNIX distributions? Or is it the case that, say as with the BSD TCP/IP stack, that they found their way into other operating systems altogether?

KAHAN: Absolutely. The library of math functions depended very little on the operating system. DEC had a good library, too. That was developed by Mary Payne and her co-workers. I think they developed it independently of ours, for the most part. Possibly in those few cases where ours happened to be better than theirs, they may have said okay, well, change it. In cases were theirs were better than ours, we were certainly happy to crib their ideas. I'm going to talk about one of them in a moment. But I know that people running different operating systems on different machines with IEEE arithmetic were using our library. It was a relatively brief period, but our library was tuned for C. It was written in C and, of course, people have other languages. They could easily copy our library, which was amply well documented within the source code. So they could easily transcribe it if need be into assembly language or into whatever other language they wanted to use. Some of our codes were written in assembly language for the VAX. Most of them were written in C. I think all of them had versions in C, if I remember rightly. That meant that if you didn't want a program in C but you wanted a program in some other language, there was no reason why that other language couldn't have a math library. Which got, as it happens, the same results. Just programmed in a different way or interfaced in a different way. I know that that was happening, it was just common knowledge that people knew that that was what they were doing. Because they could get it for free. You didn't even have to buy a book.

HAIGH: Are you aware of any commercial operating systems the code made its way into?

KAHAN: Well, I think most of them were commercial. I can't name them specifically, because we existed to broadcast the codes; we didn't get anything much in the way of flak. All I had to do was get the license for 4.3 BSD Berkeley UNIX, because that carried with it the right to use the codes, despite that the copyright belonged to the regents of the University of California. It was extremely unlikely that the regents of University of California were going to sue anyone who acknowledged their source. It said there you have to acknowledge their providence. And once you've done that well, that's science. That's what you're supposed to do. So nothing that would answer your question ever filtered down to us other than the knowledge that it was being used widely. They were selling I don't know how many copies of these tapes were being sold and they were not being sold all to universities. So that went well.

There is one special case that deserves comment. That is the trig functions sin/cos/tan. Initially, my codes for those functions would carry a version of pi that was extra-accurate. That's what I did with a calculators, and some of my friends actually carried it even further. I used a thirteen-figure value of pi. I think somebody used a twenty-digit, or eighteen- , I don't remember. Some umpteenth digits of pi beyond the precision of the arithmetic. I explained why this would be adequate for engineering purposes: it was because the identities that trig functions must satisfy would not be damaged by using slightly the wrong value of pi. This slightly wrong value of pi would effect engineering computations of things like wavelengths and antenna radiation, and so on, approximately as much as if you were measuring the distance to the moon using a laser and a corner reflector on the moon, so that that reflector sends back light in the direction from whence it came. We would get the distance to the moon wrong by something like the width of a sheet of paper, because we had approximated pi to 13 figures instead of infinity. The trouble was, however, that different places might use different extra precise values of pi. In consequence, for very large arguments the values of the trig functions would exhibit discrepancies, and the discrepancies have to be explained, or somebody has to sign off on them. They have no significance in technical computations, though they may matter to a mathematician who want to know the honest-to-God value of the cosine to the 37th, but to an engineer all you really want to do is to have appropriate undulations and make sure that when you move a certain distance the the trigonometric function changes the way it ought to. So if you go from 10 to the 7 to 10 to the 7 +1, the fact that the sine of 10 to the 7 is going to be, strictly speaking, mathematically wrong in the last few of its digits is less important than the fact the error is correlated with the error that you get at the next argument, so that the various trigonometric identities are satisfied.

HAIGH: But it would introduce the apparent inaccuracy that might be psychologically disturbing?

KAHAN: It's disturbing when you see you compare your figures to the figures of somebody on a different machines, and you discover that they disagree in surprisingly many digits even though the qualitative results are the same: the amount of energy being transferred by the antenna, the locations of the nulls or the dead spots, the types of reflections, etc. So it became clear that to lessen the burden on the people who have to answer the telephone the right thing to do is to simply calculate the trig functions right regardless of the size of the argument. That means that you have to know a lot of digits of pi. How many? Well, we were doing this independently from the folks at DEC, but they had reached the same conclusion. I suspect for the same reason. They came up with a very similar algorithm to the one that Bob Corbett came up with. Bob Corbett said, “You know, what you’d like to do to one of these big numbers is to reduce it modulo pi by 2. So you multiply it by 2 over pi, you get an integer part, of which only the last couple of bits matter, and you get a fractional part, and the fractional part you can multiply back by pi over 2 and do the usual things. So now you don’t have any trouble.

So all you’ve got to figure out is this: when you multiply your argument, which is a floating point number of a certain width, by the digits of 2 over pi (which as it happens are infinitely many), which digits of 2 over pi matter? It turns out, if the argument is very big, most of the leading digits of 2 over pi don’t matter, because they are only going to affect binary digits well to the left of the ones you care about. The digits of 2 over pi far enough to the right don’t matter, because they’re only going to affect digits of your project further to the right than you care about. So it’s just a certain section of the digits of 2 over pi that matter. It’s a section that is roughly two or three times as wide as your precision. Hey, we can do that! We can store the digits of 2 over pi, there’s a way of computing them. We took advantage of the continued fraction that had been published by Bill Gosper when he was working with MACSYMA over at MIT, but there’s a lot of ways of calculating 2 over pi. If your argument is not very big you can use the standard techniques, which work just fine. Once your argument gets bigger you can use this trickier thing that Bob Corbett came up with, but the problem is that you’ve got to store the digits of 2 over pi, and that’s a lot of digits.

How many? I came up with a scheme using Gosper’s continued faction and some other stuff to figure out how many digits we would need. It turned out that we needed half as many digits as, later, we found the folks at DEC were using. Obviously we had a niftier way of figuring out what to do and that is actually written up in the document. I wish I could find…I expect Stuart McDonald has it. Maybe I’ve got it too. It’s a document that explains how we discovered how many digits of two over pi had to be stored. The fact that there was half as many was worth knowing, because the number of digits you have to store depends on the over/underflow range. It’s really the underflow threshold. If you are going to have a very wide range of floating-point, you are going to have to store a lot of digits. The fact that we said no, only half as many as you might have thought, that really had some substantial value. That was where we got one of the letters from Vic Vyssotsky. He was astonished that we had gotten away with so few digits as we had, and he’d checked it out using very wide-precision arithmetic and he was happy to confirm that yes, we were bang on.

So we had developed a program that, I believe, is imitated worldwide and gets rid of the argument. Now you could imitate DEC’s scheme, or you could imitate ours. It’s pretty much the same schemes except that you only need half as many digits of two over pi as you might have thought. It’s very widely imitated in respectable libraries, and the crucial thing is, now we don’t have these pointless disputes. I thought that was worth…the psychological aspect of transcendental function approximation.

HAIGH: On the topic of libraries you also mentioned that the fdlibm from Sun Microsystems is maintained mostly by some ex-students of yours. I think some of whom you’ve already mentioned and some you haven’t, so perhaps that would be a good time to talk about the students who were working on that.

KAHAN: David Hough was one of my first graduate students; after Betty actually. He wrote an interesting thesis because it was not possible to solve one of these horrible problems. Remember I told you about these surfaces in the space of data and they self-intersect, and so on? It turned out that that was, combinatorially, much too daunting. So David's thesis was a somewhat negative appraisal of the prospects of finding these points reliably. We could write a program to find them, but you could never know that you'd found them all and you'd never know that you'd found the nearest one; you just know that you'd found some. That was very disappointing but, nonetheless, it was a thesis.

Hough went off to work first for Tektronix, then for Apple and, finally, for Sun, where he has been for very many years and has acquired a very broad background in scientific and engineering computation. When Sun was put together they had to have somebody to make sure they had a math library that would suit the machines they had. Now initially, they were using 68020, 68030, 68040s, etc. from Motorola, and they came with first the coprocessors 68081 and 68082, and then the 68040 at Peter Tang's library. So they had a math library, but they had to make sure that the interface of the math library was about right so they had to employ somebody who would know a little bit about that.

Later when they went on to SPARC, they switched from the Motorola chips to their own homegrown chips made by various…made by Fujitsu or whoever, by Ross Technology, I think, in Texas. When they switched to those, it meant that they would really have to build their own math library from scratch. The manager of the group to do that was and still is Greg Tarsy. Greg and I came to an understanding that they could use the math library that we had developed at Berkeley and possibly evolve from that. They might very well employ some of the guys who had done it and they do as you see the list there. Because they employ Hough and Liu and K.C. Ng part-time and Bob Corbett is there with the Fortran compiler.

HAIGH: There is also a name here we haven't talked about before, Doug Priest.

KAHAN: Doug Priest is also employed at Sun, but he wasn't initially part of that project. That came later. The understanding was then that they would put into the public domain and maintain a math library because it wasn't going to be maintained at the University of California, since the University of California divested itself of Berkeley UNIX. And they've done it; he's lived up to the terms of that agreement and more. So fdlibm is the freely distributed math library. It's posted principally on Web pages maintained by David Hough.

K.C. Ng, who had written the initial library for National and who helped rewrite the other parts, went on to a job working for an Israeli company that was interested in medical imaging. When it looked as if that company—I think it was called Daisy Systems—was going to go under, we had to find another job for him because he hadn't been there long enough to get his green card. So he got employed at Sun and got his green card, and then decided to become a Buddhist priest. I have often thought that that business of the Buddhist priest is partly my fault. K.C. was lonely and, if I had been a Japanese professor, I would not only have found him a job but I would have found him a mate. But I was not a Japanese professor and so I did not live up in fullness to this obligation, and I think it was loneliness that drove K.C. into the Buddhist priesthood. But I can't be sure and that's not exactly something I could ask him and get a straight answer. K.C. is an interesting guy, and you may think of a Buddhist priest as being somewhat…you wouldn't think of a Buddhist priest having an interesting sense of humor. K.C. does, and it's a subtle, low-key sense of humor.

The other guys there… Let’s look at Alex Zhishun Liu now. Alex did his military service in Taiwan, I think he rose to the rank of captain—well, anyway, no higher than that. But he was an officer and, I think, that may have been important when his wife and I agreed that we wished he would stop smoking. I had to find some way of doing it that would nag him and I remember asking him one day, “Alex, do you really feel okay about taking orders from that little bit of paper and weed?” And that may have been the last straw so he stopped smoking; I’m glad to know that. Alex got his master’s degree under me. Then he worked for Parlett and he got his Ph.D. thesis in something having to do with matrix computations and eigenvalues.

The work he did for me involved these test programs that I mentioned, using continued fraction techniques to generate these kernel functions and using other more devious techniques in merely continued fractions to generate them in such a way that using only arithmetic of the precision of the functions under test we generate values of those functions as expressions. If evaluated in the right order, they can be subtracted from the target function and give you a difference that is accurate to at least three more bits than your precision. So in effect, you can estimate the error in the approximation to the target function. You can estimate the error to within something of the order of a sixteenth or an eighth of a unit in the last place. That means that, if your error is sufficiently less than one unit in the last place, all your tests will show you that as long as your actual error is something of the order of 5/8 of the unit in the last place, or ¾ of the unit in the last place…no worse than that. Then these tests will then tell you that it is no worse than 7/8 of the unit in the last place, which is less than one by enough that it’s a safe margin.

These tests were extremely devious and Alex is a good programmer of the extremely devious. So we used his tests first on the Berkeley UNIX codes, and then they became canonized as part of the UBC test, part of a bigger test package. This was also maintained at Sun, maintained by David Hough and the other guys, and was also widely distributed and very widely used. Alex would receive testimonials; I think I’ve said he could paper his bathroom walls with testimonials from guys not only about having checked out their functions but having checked out their compilers and their hardware, too. Because if a discrepancy arose, who could say where the discrepancy originated? It might not be in the transcendental function code, it may be a bug in something else. So as David Hough has said frequently, those tests are tests of the whole system. They don’t just test the transcendental function codes.

Other tests were done by later students like Mike Parks. I don’t know if Mike Parks’ name is there…

HAIGH: No.

KAHAN: Oh, it should be. We’ll get to it. So let’s see, that was Alex Liu. Stuart McDonald was a free spirit who really didn’t want to do what he didn’t want to do—very Californian. Interested more in number theory than in other things. He accreted to this project and he helped us with the tests, particularly the code for the ascertaining of how many two over pi we need. I don’t know where he is now.

[Tape 12, Side A]

HAIGH: You were discussing your students, particularly those involved with Sun Microsystems.

KAHAN: With the 4.3 BSD Berkeley UNIX library. We talked about Alex Liu.

HAIGH: You haven’t said anything, I think, about Doug Priest.

KAHAN: Okay. Doug Priest was actually Steve Smale's student, forwarded to him by Lenore Blume. His task, initially, was to explore ideas of Steve Smale and Mike Shub about a complexity theory appropriate for real computation. They had a seminar going and their idea was: if a precision of an arithmetic was limited, then the accuracy of results must be limited in some way. What was the maximum accuracy you could hope to get if your precision is limited in this certain fashion? That would, in some way, characterize the difficulty in one aspect of the complexity of computing some real function. I gave a talk in a seminar in which I said there is no limit of an accuracy derived from precision. If there is a limit it is limited by the over/underflow thresholds. The reason is that floating-point arithmetic follows rules; on every computer it follows rules. On IEEE arithmetic it follows particularly favorable rules. Therefore it doesn't make sense to discuss the limitation on accuracy imposed by precision. Rather, it's a trade-off: how much accuracy do you want? It will cost you more time.

So this somewhat derailed the direction of Doug Priest's thesis researches, and he started to work on the direction I just enunciated. So in effect I became his thesis advisor even though Steve Smale was the first signature on his thesis. He did indeed explore the possibilities and he provided constructions whereby you could, in principle, achieve arbitrarily high precision—limited only by the underflow threshold, if you are willing to wait. You could estimate how long you'll have to wait and so on. It isn't if you had to wait forever or anything. His ideas ultimately spread around. He published his initial version of his thesis in one of the IEEE sponsored computer arithmetic conferences. [Douglas M. Priest, "Algorithms for arbitrary precision floating point arithmetic," 10th IEEE Symposium on Computer Arithmetic, 1991, pp. 132-143].

Ultimately it influenced Jonathan Shewchuk. He is now one of my colleagues in the computer science department. Shewchuk came to the conclusion that geometrical computations, which occasionally become inconsistent because of round-off, thereby cause programs actually to crash or, at the very least, produce extremely weird-looking pictures. He came to the conclusion that in the end the only decent way to deal with these things was to use extra precision on those occasions where it was necessary. So he developed from Doug Priest's stuff, he went further and developed schemes which would be more economical in certain cases where we were doing geometrical computations and you know in advance what you have in mind. Typically these will be geometrically computations of determinants. Computations of determinants involve only adds, subtracts, and multiplies so you don't need the full panoply of stuff. But Shewchuk now has programs of extraordinarily wide interest. In fact I think his website, which was originally of Carnegie-Mellon where he got his degree, gets perhaps more hits than almost any other, except pornographic.

Priest now works for Sun and he contributes to the numerics group. I think he works on other things; I don't know exactly what he works on. We don't see each other very often. He is a very quiet fellow and a little bit like David Wheeler. It's very hard to get him to talk.

HAIGH: You might also want to say something about James Demmel.

KAHAN: Jim Demmel is one of my star students. I mentioned Jerome Coonen who was for a while one of my two best students; now it's three. Demmel is a hyperactive child and I don't know where he gets the energy to do all the things he does. He wrote a beautiful thesis—initially an attempt to see what could be done with the ideas that I expressed in my paper on conserving confluence. What he discovered very soon was that the combinatorial problems where formidable. What it really means is that the intersections and self-intersections and so on form a kind of froth where, in general, it’s extraordinarily difficult to figure out which of these, let’s say “most self-intersected points”… it’s very difficult to determine which one is nearby.

It explained why Hough had problems. But he didn’t prove that this is the case. The proof came later from one of my young colleagues named Ming Gu. Ming Gu actually proved one of Demmel’s conjectures about the exponential growth in complexity in work involved in trying to deal with one of these problems adequately. But what Demmel did was to show that you could estimate condition numbers; you could estimate hypersensitivity by using appropriate techniques, and that the estimates of how sensitive a problem was to perturbation would cost you at least as much as computing solution, if not more. He had various algorithms, which came to be incorporated with other work reasonably widely.

Algorithms for discovering what’s called the Jordan normal form, or the Jordan structure of a matrix also work on more complicated, what are called canonical forms. He collaborated with me on papers designed to show how you could, in some cases, compute results that were as accurate as the data deserved, instead of being merely as accurate as the conventional backward error analyses allowed. That, in it’s own right, has turned into a major industry. There are people doing these things all over the place. They discovered ways to get good answers which are good in an intuitive sense instead of merely good in some error analyst’s norm.

Demmel has also been the technical ramrod for the CITRIS Project here at Berkeley, which attracted some $300 million. Unexpectedly, since I know the state legislature had intended initially not to send money to Berkeley, feeling that Berkeley had more than enough money and it was a pain in the neck, anyway. The proposal that Demmel helped assemble from various sources and so on was so much better than all the other proposals that they had to lay on an extra section, so to speak. Its like an extra pile of money. He is also been a major organizer of the LAPACK project, the extended BLAS, things like that. The most recent accomplishment has been a program intended for LAPACK and ScaLAPACK to do iterative refinement, which means that when you want to solve a set of linear equations, there are various things that can cause the accuracy of your result to be less satisfactory than it should be. Even though the residual may be small, the accuracy may be poor. But iterative refinement is a way of overcoming these difficulties without having to figure out why it went wrong. It was a scheme that had been analyzed by a number of people.

For example, Cleve Moler, when he was a student (and I knew him at Stanford in 1966) actually wrote a paper on the subject. A guy named Robert Skeel, a Canadian, who I think may be at Illinois now. He wrote a paper on these but there papers indicated that if you used iterative refinement, you could reduce your residual. But the trouble is you can reduce the residual while you make the solution worse. What I instigated by writing examples in MATLAB and then that turned into this project was that, if you can compute the residual extra-precisely, then when you do iterative refinement you can actually improve the solution.

What with various refinements added by Jim and by the students that we’ve been working with, we can not only refine the solution, we can even give you a reasonably good realistic overestimate in the error in the solution so that, under most circumstances, people would have a solution as accurate to within a few units in the last place. If you don’t like the accuracy, it’s because your data is grubby. It’s not because the linear equation solver has done anything. That’s quite an improvement and doesn’t cost much in normal cases, too. Whatever people had been satisfied with after Wilkinson’s analysis and mine too, whatever they had been satisfied with before. And currently, you can get a really good solution instead of merely one with a smaller residual. It doesn't cost a hell of a lot more than what you used to pay.

Demmel is a marvel. He went off to a post-doc in Germany, and came back and worked as an assistant professor at NYU. I remember in my letter of recommendation for him I said, "You can have him for a while, but I'm going to want him back." So we did get him back, and my colleagues are all delighted with that decision. He is now a very important member of the department. We appointed later in computer science, Cathy Yellick, who is interested in programming languages in parallelism. I remember thinking at the time—but I won't express my thoughts, I had to bite my tongue. Anyway, they got married and that was what I pretty well figured would happen. They live just a couple of blocks away; they have a couple of kids. Cathy now has, I think temporarily, a position up at LBL, which is running simultaneously with a position down on campus. I hope that that means that they're planted here as permanently as I am because I would be distraught if they left.

HAIGH: All right, so I'm seeing the names of three remaining students here that you have not discussed. I just marked those on the list.

KAHAN: Yes, right. Ren-Cang Li came from a tiny town in China—so tiny, that the Chinese did not think it worth their while to post a garrison there during the Second World War. He got a scholarship and was the first member of his family to go more than 100 miles or so from their village. He got his bachelor's degree in Beijing and was accepted as a graduate student at the University of Illinois at Chicago. He then wrote a letter asking to be accepted as a student here, because he couldn't find anybody there who was really up to what he wanted to do. He sent a couple of examples of his work and I thought, good God, this guy has already written more than most professors—and hard stuff, my kind of stuff. So I wanted to bring him to Berkeley, but it was an unfortunate time because the dean of the graduate division was angry with the math department for having lost track of several of its graduate students. He said you can't admit any more graduate students until you clean up your system and figure out what has happened to some of these guys. That was just when I wanted to have Ren-Cang Li come. So I had to write a letter and they did relent. He turned up I think in January of 1991, I can't remember, and he aced the math department's prelim exams. He stood first on the prelim exams, which vindicated my judgment.

Ren-Cang has worked on so many things that I've lost track. He was writing programs for the LAPACK project and programs to help Parlett. This was in matrix computations. He was writing papers on matrix error analysis, and that seemed to be his field; I thought it would be a good idea to have him try something else so I got him to join me in some of my work in unconventional methods for solving differential equations. His thesis involved some algebraic computations to get more coefficients in the-- there is a series of expansion associated with the names of Hausdorff and Baker. So he used the algebraic system MAPLE on an IBM Iris 6000 with an awful lot of memory in order to compute some more of those coefficients to fix errors in previously computed coefficients, to compute, then, coefficients for enhancing the accuracy of some of these unconventional schemes (which are very close to my heart. These are things I really would rather be working on than the IEEE standard) and discussing the stability. It's a very good thesis.

We've got a couple of papers out there. One of them I think I co-authored with him, and there's another really interesting paper on this subject. He felt it necessary for the sake of financial security to accept a position at the University of Kentucky in Lexington. His English was not sufficiently good that I could, with a clear conscience, send him off to a place like Caltech or UCLA or Stanford, which is really the level of institution that he ought to be at. But his English is sometimes difficult. Now it's not as bad as some of my colleagues, but then I wouldn't have recommended some of my colleagues, either, on those grounds.

I'm also an advisor and students come and tell me that they can't understand the professor. They complain about one of my colleagues, who sounds like a chipmunk. It doesn't seem to me that we ought to do that. We ought to compel some people to take courses in English as a second language, focusing on pronunciation. But you don't want to get me off on that right now; you don't have time for it.

For a while, Ren-Cang was at the University of Kentucky—you could say, perhaps, resigned to his fate but, nonetheless, taking part in conferences and writing papers as well as handling students. He came and visited us for a while and we took him to one of the markets nearby which caters to Asian clientele and I could see that there were tears in his eyes. He would really rather live in the Bay Area.

So we got him a job at Hewlett-Packard. He took a leave of absence and he worked at HP on their numerical stuff for a while. But then Carly Fiorina got into own of her cutting modes and insisted that people be cut in order of seniority, so that group had to let Ren-Cang go. Although every other member of the group said he was, by far, the wrong one to let go. Even while at HP he had been writing interesting papers and solving interesting and difficult problems.

Now he's back in Kentucky. He's got a wife and daughter and, I think at this moment, he's in China dealing with some family matters. I regard him as one of my three best students and I would hate to have to say which, Ren-Cang or Jim Demmel or Jerome Coonen, would be the best because that would be like comparing oranges and apples and roast beef. You just can't do that.

Who else have we got here? Scott Baden is now a professor at the University of California in San Diego. Scott was originally interested in computer architecture and was going to work under the supervision of Alvin Despain, when Despain was a professor here. Despain is now at Southern California, where they are paying him more. Scott was assigned the task of figuring out what would be the kind of architecture that would ease the problems of programming that takes advantage of concurrency and parallelism—in some cases on a massive scale. What should architecture look like? Scott came to the conclusion that the architecture wasn't really the issue but it was the programming tools and the programming support. Alan Despain said he didn't really want to supervise a thesis in philosophy. So Scott came to me and I said I'm not used to supervising a thesis in philosophy, but if you can demonstrate a concrete instance…but to demonstrate a concrete instance, he would have to take a course in numerical analysis involving partial differential equations where power of computation is essential.

So he did and, within a few months, he demonstrated that yes, his ideas actually made sense. They provided an interface to certain facilities, which can be architected in very many different ways. But given these facilities, the applications programmer need only invoke these facilities in his program and then his program will run on whichever of these architectures involved. That was really a good job so I was happy to recommend him. There he is now, a flourishing professor at U.C. San Diego.

Doug Greer—now, he had a really interesting situation. Doug was employed as a programmer by the guy at the U.C. Medical Center in San Francisco, where they were interested in what you could find out about the brain if you planted electrodes on the skull, or if you used magnetic resonance imaging, or things like that. This might not seem like my cup of tea, but Grier couldn't find anybody to supervise him so I took him on. Doug wrote an interesting thesis. He had written several papers, some of which had really interesting pictures if MRI scans using cross-sections of the brain inside the skull. Of course now that stuff is relatively routine. At the time, which was 1990 plus or minus a few years, this was cutting-edge stuff. Particularly difficult was the problem of resolution, because the MRI scans had to be spaced a certain distance apart or else they would just sort of blur one into another. What he did was to find a way of reconciling three families of scans. There are horizontal scans, there are vertical scans in one family of parallel planes, and then vertical scans in an orthogonal family of parallel planes. Naturally, these scans are not altogether consistent, because there are distortions. His thesis would reconcile these distortions in an interesting, non-linear way. That was a pretty good thesis. And I still hear from Doug Greer from time to time.

There's not….Mike Parks here. Well, Mike Parks is really one of Beresford Parlett's students. But he finished his thesis at a year in the '80s when there was a dearth of jobs. That year, I think something like half the math department's graduates of Ph.D.s did not have jobs as late as August. So this was somewhat of an emergency situation. What were they going to do? Some of them had sent out 100 resumes—no nibbles. So I said what I'm going to do is set up a course on floating-point arithmetic. A one-day course in floating-point arithmetic. If these kids take that course they might find employment in the computing industry where people who know anything about floating-point arithmetic are very scarce.

Mike Parks had been working for me in the meantime—because he didn't have anything else to do—on some problem, which is essentially adapting Brian Smith's program to current languages. That's kind of a zero finding program. But he didn't finish that work because he got a job. He got a job on the strength of attendance at this one-day seminar and got a job in Austin, Texas working for AMD. Now Mike had been the guy to organize the logistics. Get the room, send out notices, and stuff like that. Unfortunately, the math grad students for whom I laid this on didn't come. Instead we got people from Texas and Georgia and all sorts of places, because they had seen on the Web the advertisement for the free, one-day seminar on floating-point. So the room was full. It was full to overflowing, and it's a room that would have held about as many as 30 people—and we had 40 in the room. But we didn't get the graduate students. Well, we got Mike Parks. So Mike Parks got that job, and then AMD, as was its wont, would fall upon hard times and would cut people, again according to seniority. So Mike ended up with a job with Sun, where he still works.

HAIGH: Now, in an unpublished document from you I saw the phrase, "My students know that their work belongs to them and they belong to me." Will you explain that?

KAHAN: That's right. It isn't obvious? Well, I'm very jealous about my students' work and I insist that they should publish well enough that they will get credit for the work they've done even if the work came from things that I said they would find worth doing. Once they have understood a problem and its solution, and how to explain it, the work is theirs. And so my contribution becomes a footnote. They publish the paper. I'm not one of these guys who insist that I should be listed as a co-author on anything my students have written on what they've done in engagement with me.

I will not employ students as slave labor. For example, unlike a professor I could name, I don't do the math and they do the programming. No, they are going to get a rounded education. If I want them to work on some task as students, they will work on it because I'm convinced it will contribute to their education. And once they've gotten as much as they can get of their education from it, if it isn't quite finished or it isn't quite perfect, that doesn't matter—because remember what my job is: it's to produce people.

But once they're produced and they are out there, they belong to me. Okay, they are my students or my ex-students and we maintain a relationship. I think I still have a good relationship with all practically all my students. There is one who has fallen into a black hole. I don't know where he is; he was a student at Toronto, and I don't know where he is gone. But all the others, I maintain a relationship with them and they acknowledge it. I rejoice at their subsequent successes.

HAIGH: In the remaining time it seems there are two current projects we should talk about at least briefly. The mixed-precision BLAS and the revision that the IEEE standard set--

KAHAN: Okay, mixed-precision BLAS. The idea is that, if you can either in hardware or simulated in software compute some things to extra precision, perhaps at a higher price than you'd like to pay, then what can you do to enhance the quality of computation? I've mentioned iterative refinement; it's a very good example. All you have to do is compute a residual extra precisely. That amounts to, in most cases, a matrix multiplier or two. If you can do that the, because cancellation occurs fairly substantially in a residual, you then have a result which does not have to be retained to extra high precision, but can participate in ordinary precision, work with things you've already computed in order to enhance the quality of your solution. That is one of the things that is going to come out imminently in a paper by I forget how many-- it's got a long list of student authors, and I'm in there somewhere, too.

HAIGH: What is your personal contribution to the project been?

KAHAN: Well, initially it was to start it going and to say, "Here are the algorithms I'd like you to look at and this is the type of error bound I'd like you to get." As things got going well enough and I got ensnared in this IEEE thing, they kept on doing the work. Indeed, they are doing it better than I might have. So I participate from time to time and supply some test examples here and some hard cases. Unfortunately, I haven't done as much writing as I should, because it takes me an hour to write a sentence. There are things I wish I had been able to write out, like what are the continuing failure modes? Because there are failures—software has failure modes and this scheme has its failure modes. It's got its limits. I'm the right person to write that up and I just haven't had the time.

HAIGH: So is the idea that this will be a drop-in replacement for the existing level three BLAS that will just provide these extra capabilities if people want them?

KAHAN: Yes, you can use it for a replacement for some of the level three BLAS, and for enhanced level three BLAS in other cases.

HAIGH: Does the need for this have any relation to architectural developments in computer hardware?

KAHAN: Yes, because the hardware supports some extra-precision, like Intel's 10-byte wide (although you store it in twelve or sixteen bytes, but Intel's 10-byte wide format is a natural one to use when computing residuals). But of course that works only if you use the stack. If you use the multimedia instructions, you won't get that extra capability. Then on the other hand, IBM's binary floating point has a quadruple-precision format which runs somewhat slow, but not so slow that you can't afford to use it for this purpose. So indeed the paper that is coming out is just perfect for IBM. It's just ideal. You do your computation in double-precision and you say, in order to clean it up, I'm going to compute the residual in quadruple. That's just a drop in the bucket as far as the time is concerned. And then you do iterative refinement, which is just another drop in the bucket. And now you've got, in almost all cases, a very much enhanced solution, so it's much more reliable. On Itanium, I think you get the same benefit. The same kind of thing, although in Itanium, the quadruple-precision is in software. The same thing could be said for the Sun SPARCs.

HAIGH: Okay, how about the IEEE standard revision?

KAHAN: Yes, well that has been a bleeding sore for a long time. We've gotten distracted because we're discussing decimal arithmetic. I know that I warned initially that the right thing to do would be to finish binary and then work on decimal. But we got persuaded (perhaps against our better judgment) that we should include decimal at the same time, and that has turned into an enormous distraction. It's a distraction partly because it's very difficult to describe the way that decimal arithmetic works; it combines fixed-point as well as floating-point arithmetic within the same encoding. Trying to describe both simultaneously is, I think, a mistake. If we weren't sitting here at this microphone, I would be trying to write up exactly how you should describe these things so that people who wish to segregate them in their minds can do so.

HAIGH: When did work on the revision start?

KAHAN: It began I think in 2000, or maybe it was 1999. It was probably 2000.

HAIGH: What was the initial impetus for the project?

KAHAN: Well, every IEEE standard is supposed to be reviewed, I think, every five years. They hadn't really given this IEEE 754 a review of any consequence. Finally, it seemed like a good idea to do it in the year 2000. The thought was it would only take a couple of years. You dare not change much of this thing. But, oh boy, there are people who love to make some changes because they have forgotten, or perhaps never learned, how to do the whole thing. They never attended this long-running seminar that started in 1977 and ran on until 1985. Well, actually until 1981.

HAIGH: So is the committee process more formalized now than compared with the open meetings of before?

KAHAN: Well, I don't think so. If anything, it is a bit more ramshackle than before. We have a meeting a month with the whole committee, and a bi-weekly meeting of a subcommittee trying to actually grasp the wording. And unfortunately, some of us have daytime jobs and can't…. You see, when I worked on the IEEE standard, I spent a good deal of that time on paid industrial leave being paid as a consultant. I was earning rather more money as a consultant than the university had been paying me, so it wasn't a discomfort for my family or anything. But now we don't have that arrangement, so I can't work on the thing full time. I don't have a Jerome Coonen now. David Hough is doing the best he can but he, too, has a daytime job. The chairman, Dan Zuras, has retired from HP, but he is not the best person to write something up. So our progress is not all that great. But I have some good people on that committee, not all of them ex-students. I'm hoping we'll be able to see the light at the end of the tunnel, but I mustn't say when that will happen because that would be like casting the evil eye on it.

HAIGH: I know one of the things you are hoping to get into the new standard is better exception handling and software support.

KAHAN: Absolutely, yes.

HAIGH: Are there any compiler people involved with the committee?

KAHAN: That is part of our trouble. We do have compiler people. Jim Thomas, who is an ex-student, he got a master's degree and went on to Apple on the strength of the understanding of floating-point and, particularly, the defects in Apple III and Apple II. Now he works for HP and has been a major force in the incorporation of numerical capabilities aligned with the standard into C99. Jim Thomas contributes to the committee by visiting maybe once or twice a month. He has a full time job at HP. He is certainly knowledgeable—he has become knowledgeable about languages. Joe Darcy, an ex-master's student who worked on the paper, *How Java's Floating-Point Hurts Everyone Everywhere*, is currently Sun's floating-point czar in the Java group. He also contributes, but he too, has a daytime job and that's part of the trouble.

But we also have some others. We have a guy who is part of a little start-up company and he prides himself on the fact that he never did pass a math class. He is a programming language expert, I think initially on the strength of the fact that in high school he took a number of languages. Can I remember them all? There was Greek, Latin, Germany, French, Spanish, and English, or something like that. But unfortunately, he doesn't know anything about numerics so, although he interferes where languages are concerned, he doesn't have any grasp of why the numerical requirements are different than, say, operating system requirements.

No, we don't have enough help and although I canvas my colleagues and see if there are people who would help, it is extremely difficult to get them. Just recently we got a couple of weeks of attention from a graduate student who is just finishing a thesis. His name is Wes Wymer and he wrote a thesis on exception handling difficulties, about exception handling problems were those tuned to operating system exceptions. Operating system exceptions are not like floating-point exceptions, which are much more lightweight. Nonetheless, he was able to give us some advice on how to draft things. But I think you've put your finger on the problem. We just don't get adequate interaction with the language community. They're just not interested and I don't know how we are going to deal with that. But we are going to deal with it, one way or another we are going to deal with it.

HAIGH: Is it your expectation that the new hardware standard will be widely incorporated by manufacturers?

KAHAN: It's a misconception that IEEE 754 is a hardware standard. It starts off by saying that is standardizes the ambience for programming. It has been interpreted as a hardware standard. Unfortunately, that means that the hardware guys did their job and everyone else thought they didn't have to do anything. But that was a misinterpretation.

HAIGH: With the new standard then there will still be a lot of work to do to bring the software onboard—that will happen?

KAHAN: Absolutely.

HAIGH: Are you at least confident that the hardware people will adopt the new standard? They won't just stick with what's already been done?

KAHAN: Well, it's worse than that. If the duty cycle on some of the features of the standard turns out to be low because the languages refuse to grant programmers access, then some of the features of the standard will atrophy. We can see that happening now, and have to fight against it. There is a certain desperation about this process. We really do want to get the language people to support things and then, of course, there is a period of education where we can get the applications programmers to realize what they can do. It’s not so easy because if a task is sufficiently difficult to be deemed impossible, then nobody tries to do it. If we make it possible and practical, that doesn’t mean that folks are going to jump on the bandwagon and say we can now do this horrible thing that previously we couldn’t do. It’s horrible. Dealing with exceptions and making software robust means you have to deal with all the corner cases and the loopholes and so on. And there are so many. I’m trying to reduce that to something that is humanly manageable. The alternative is unthinkable. Let’s suppose that computers are running at tens and hundreds of megaflops, you get some sort of anomaly that bugs you every few months. So every few months you have to spend a few days trying to figure out what the hell is going on. Maybe you can persuade someone to change your data or you can trick your code or something. So you have to do it every few months and that’s the way you earn your living.

But now we are running at gigaflops and even tens of gigaflops. So what used to happen every few months is going to happen every few hours. How are you going to cope? We’ve got to deal with the situation, so that’s one of the reasons why I feel it is a desperate necessity.

HAIGH: You’ve been similarly involved in a number of areas writing papers exposing, in a very detailed kind of way with arguments and deficiencies in numerical computation in Java, in the Cray arithmetic, problems with loss of accuracy in MATLAB--

KAHAN: That’s the old Cray arithmetic. You have to understand that that was 1990. Cray has changed. It has changed hands and it has changed its arithmetic.

HAIGH: I think in all three cases, the papers really speak for themselves in terms of the argumentation and support. I was wondering if you think that, in either of the three cases, that the work and effort and thoroughness of the argument that you made had an effect on changing any of those things?

KAHAN: The effect is of course very slow. It’s slow mainly because the language community on the whole is indifferent to the problem. They’re indifferent to the problem because the language community is populated by people who have computer science degrees without a necessity and substantial exposure to a competent course in numerical analysis. Furthermore, the programming community as a whole has the same difficultly and the same limited education. It’s not altogether clear that they would appreciate what I would like to do for them. Maybe they would be happier if I just went away, who knows. If they don’t have to deal with the problem, that is best isn’t it? The problem of exceptions is one that you can’t just disregard. But the problem of accuracy—you can deal with that by making high precision fast enough. So one of the things that I urge and we urge in the standard and in the standardized quadruple precision is that, if quadruple-precision runs fast enough, then round off-induced anomalies will diminish in frequency (a frequency which, at present, we don’t know because we don’t keep score. Most of them don’t even know if the result is right or wrong).

And in any event, most of the results don’t matter because after all, they are just playing games, and if something that flickers here and there then-- There is the IBM Cell machine. I’ve written that the IBM Cell machine is justified it’s somewhat grotty arithmetic by being not as bad as it could have been, but it’s a little bit grotty. They’ve justified that on the grounds that the occasional lapse in mathematical integrity merely causes some sort of flick on the scene, then it’s gone. Then IBM announces that, once they get this thing going because it has such ferocious performance, they look forward to incorporating them in supercomputers and medical imaging.

Isn't that just great? Now you run a certain risk that, as they refine medical imaging in order to have a better idea, because medical imaging is a very noisy situation. You've got to pull things and analyze. They don't really make sharp pictures to distinguish one organ from another. You are running the risk that, on the one hand, somebody may seem to see spots in your gut and say, "You know, we should open you up and take a look." Maybe they'll discover that the spots are artifacts of a new program that thought it was going to make things clearer…but doesn't. Actually you run an even greater risk if they don't see spots. There is something there, but it was smoothed out. It was just regarded as an artifact and got smoothed out. Now you run a really interesting risk, don't you?

We've got to find a way of getting round-off and other kinds of numerical uncertainty to abate below any level that matters to us. One way of doing that is to, of course, employ error analysts. If only you can find them and there is no guarantee that if you did employ them you would be better off. But the other way is just somebody to do the computations in such extravagant precision beyond anything you imagined to be needed, that the probability of being bugged by this sort of stuff on rare occasions simply dwindles. So that's what I'm advocating.

So it's going to put a lot of guys like me out of work, perhaps; that would be okay, too. After all, I have to retire sometime. So, yes, I think there are a number of fronts in which we have to conduct this war and I may not win on all of them; but I know if we lose on any one of them, we are doomed. So I'm just fighting a battle to prevent things from getting worse.

HAIGH: It seems that the article on Java has been widely discussed?

KAHAN: Oh, yes.

HAIGH: Do you know if that has led to any concrete efforts to improve the situation?

KAHAN: The most important concrete effort was to have Joe Darcy hired into that group so things wouldn't get worse. Someday, maybe, they'll get better.

HAIGH: In the case of MATLAB you wrote, "I wish the makers of MATLAB recalling their roots would rededicate attention to their numerical underpinnings." Beyond the specific issues you write about right there, what is your general feeling about the way that MATLAB has evolved--?

KAHAN: The folks at MATLAB do fix things from time to time. And they accept repairs if someone submits them. I think they're more conscientious about this than, say, Microsoft. But they are limited by what Microsoft does because they have been using Microsoft's compilers. Maybe they should switch to Borland's compilers or someone else's. I was hoping today to have MATLAB 7 on one of my computers, but as I told you, this guy isn't coming in today. So I'll have to see if MATLAB 7 to see how much of the numerically bad stuff has been improved. I fear that in one respect MATLAB 7 is worse. From what I can tell, they are not using the 8087 descendent. They are using the multimedia arithmetic because it runs faster and, therefore, the ability to do iterative refinement is probably crippled in MATLAB 7 but I'm not 100% sure of that; I'll have to check it out. MATLAB 7 does allow you to store four-byte variables instead of always eight-byte. That has some interesting possibilities for the development and testing of algorithms, but I firmly believe that what I've written could be considered proof as much as is needed. The best way to deal with the precision problem is to design hardware that runs quadruple-precision almost as fast as double so that people would use quadruple routinely.

HAIGH: There is a phrase from your John von Neumann lecture in 1997, and I'll read that and let you comment on it: "Mathematics is a miraculous reward for penetrating thought. To render that kind of thought ever more economically was the computer's most worthwhile promise. We had better not entrust it entirely to people antipathetic to mathematical thought or motivated too much by mere pecuniary rewards."

KAHAN: Yes. Doesn't that speak for itself?

HAIGH: I think it does, but it also seems resonant with your own career and the kinds of choices that you've made.

KAHAN: Yes. Well, you certainly can't call me greedy because I'm paid as a professor at a very modest rate. I suppose if I worked for IBM I would be paid at least four times as much, by now. However, decisions like Bill Gates's decision that afflicted the 8087 and others, are frequently motivated more by greed than by anything else. In fact, I think Bill Gates, Jr. is credited with the observation-- when he was told about somebody's restraint in the marketplace, he sneered off, "Finite greed," as if that were a character defect.

The trouble is, of course, that we don't have enough of what might be called *vertical integration*, so people can realize that the trouble they save themselves is causing trouble for someone further up the line. In the computing industry, there is an intensely layered structure of dependence. From the ultimate user of the computer down to the designer of the hardware, there can be easily several dozen layers. But there are going to be at least five layers, roughly. There's logic design—the circuits, the fabrication, the manufacturer, the operating system—which is layered itself, and then the programming language and the applications programmer. You might merge some of these layers, but the fact is, there are lots of them. If you have flaws in any of these layers it induces at the very least distortions, if not failures, in the layers above it. There is no doubt in my mind that one of these layers involves numerical computations that figures in people's lives much more than they're aware.

I don't necessarily want to get them preoccupied with these things. The whole point of civilization is that we depend on other people to do their jobs so that we don't have to worry about them. You turn on the tap here and the water comes out and you can drink it. When you're finished with it disappears into various holes and is out of sight and out of mind. That doesn't happen everywhere does it? Aside from the fact that there are places where the water that comes out of the tap is definitely not for drinking, and aside from the fact that there are places where you can tell from the smell that a certain amount of water has been disposed of but is not altogether out of sight and out of mind, there are even places where people can't get the water without walking for some miles. We shouldn't take these things for granted. Somebody built the system in our civilization upon which we depend and which we benefit and we owe them a debt. The way we repay that debt, and I can't think of any other way I can think of to repay the debt, is to keep that spirit in mind that the things we build are part of a civilization structure upon which our descendants, our spiritual descendants not just my grandchildren, will depend. We would like that to be robust and be able to depend upon it and not worry. On the other hand, I think they have an obligation to learn something about it and that means they have to learn something in history beyond merely who fought and won which war.[13]

[13] The rest of the material given in answer to this question has been moved from earlier in the interview to improve the topical flow.

I'm still trying to get time to solve what are called matrix Riccati differential equations. The solution of the differential equation is a matrix, not necessarily square. It has some significance in control theory problems, some stochastic differential equations. It has quite a few applications, especially when you are interested in the eigenvalues and eigenvectors of a matrix whose elements are functions of a parameter determined by solving differential equations. The trouble with matrix Riccati equation is that the solution can have poles—that is, it can go off into infinity. If you think of it as an analytic function, it's like function one over *x* where, once you get past the infinity, you just continue in another branch of the graph. The graph is a hyperbola, it just happens, that it goes off to infinity somewhere. And this is analogous to the matrix Riccati equation—the matrix goes off to infinity somewhere, but it comes back if you keep on going. The trouble is that numerical methods are inchworm methods for solving differential equations, so they creep up the graph. There is a pole here and it is creeping up the graph and trying to climb up and up and up, and it can't get up and over the pole because it never gets to the top. It's stuck—and my numerical method has a number of delightful properties. There are group theoretic properties, which I won't go into, but the important property is this: as long as you don't step on the pole, which is a land mine, and step over the pole, you will find yourself on the correct side of the graph: the one coming back up. It's as if the pole wasn't there, as far as the numerical accuracy is concerned. There isn't anybody—nobody else has numerical methods that do things like this. The only thing you have to do is make sure you don't actually step on the pole because then you blow up. You get an infinity and, from then on, matrix applications with infinites is a dead loss.

So the only thing that remains is to find really reliable ways of discovering not so much that you are approaching a pole as where the pole is accurately enough, so you'll know to step over it instead of stepping on it. That's what I wanted to do with my sabbatical. I had ideas of how to do that; I think I can figure it out, it looks right. I just haven't had time to do because of this damn committee. That's the sort of thing I would prefer to be doing with my time.

But if I can't get the arithmetic to work right, then it gets very, very difficult to solve these differential equations. Remember I mentioned differential equations with singularities but regular solutions? This is another example of a situation where if the arithmetic malfunctions—if it doesn't do the things that decently designed arithmetic can and should do, or if it does it too slowly—then I can't get these methods to work. For that matter, the same thing happens in massive matrix computations, where there are vast numbers of multiplies and other things going on, and there are a few little places where you have to do some things slightly differently. If you can't get the arithmetic to work properly and do what it's supposed to do and to do it quickly, then you go nuts trying to figure your way and fight your way around this arithmetic instruction. These things shouldn't be there.

It's one thing to find them there for old geezers like me, because there was a time when we old guys used to take pride in finding tricks that would get around these obstacles. That was part of our craft. At SCIDS, for example, we would go and boast about how we had done these marvelous little things. I don't want to inflict that upon young people in the next generation. We shouldn't teach them these miserable tricks occasioned by dumb arithmetic. I would like them to spend their time usefully.

In any event, we aren't going to have many of these guys because the educational system is somewhat screwed up. So we are going to have a shortage of these people and, therefore, we've got to conserve carefully the energy and the time of the ones we're going to get. We can't have them dissipating their time on these dumb things that my generation was so proud of dealing with. If we are going to be civilized, we have to arrange that the younger people benefit from our experience without having to relive it. There I am, stuck having to do these miserable things at my advanced age, working on this IEEE committee, trying to fight through exception handling, and so on. I think all these things are obvious. What has to be done is perfectly obvious. What's not so obvious is how to persuade the folks that need persuading, because they are not going to be the immediate beneficiaries. The immediate beneficiaries are sometimes two or three removed. It is notoriously difficult to get one person to confer a benefit upon another that he has never met.

HAIGH: That seems to lead us to the question I wanted to raise of the importance of the understanding of the history within the computing field.

KAHAN: An enormous amount of what we do in computing is an arbitrary accident of history. The nature of programming languages depends a great deal upon these accidents—the fact that we have to stay compatible with past practice. Think of all the distortions introduced into Windows because it had to rely upon DOS. That's part of the reason for part of its unmanageableness now. If we don't learn something about the history of our technology, we won't realize which things are accidental and which things are essential. If you come to think that everything is essential, you are paralyzed. If you come to think that it's on accident so I can be as creative as I want, then what you are doing is ruining some of the delicate structures of the past that have actually been proved necessary but, because you didn't learn any history, you didn't realize that they were necessary.

I told you that the IEEE standard stuff is essentially, mathematically necessary. There are very small variations, things could be slightly different; but for the most part, it's built on rails built by mathematics. If you decide to throw away parts of it because they're inconvenient for you, that's okay, if you are throwing away only for stuff that is going to matter only to you. But if it's going to matter to someone else, you can't be too sure that, perhaps, you have violated laws of mathematics and mother mathematics is going to punish somebody. I wish it would punish the person that violated it instead of the one who uses these rules.

So you've got to learn some history to affect the distinctions. But we aren't teaching that history, we just don't seem to have the time for it. Worse, we don't seem to have the time even to maintain history. In my mathematics department, we don't have the room to keep my departed colleagues papers. We are going to pay a price for that, and I don't think the price is going to be so simple as some of the things by [George] Santayana. You know, people who cannot remember the past are doomed to relive it. I don't think it's going to be that simple. I think we're going to find that the folks who don't understand enough of their past are going to create impasses—they are going to create situations where they just can't escape. I see that, for example, in water. Water is getting very scarce in California. People don't remember how the decisions were made about zoning and population densities and things like that. They think they can just build anywhere.

But we're running short of water. What happens when they really discover that a shortage of water is endemic and conservation isn't good enough. Are we going to go to war? Perhaps it will decide that there is a lot of water flowing north in Canada, and it just ends up in the Arctic. Why don't we build a great big pipeline to bring the water down here? First, the Canadians may say, "Sure you can build a pipeline, but it's going to cost you something." If the Americans say, "No, that's too much to pay. I'm accustomed to paying this much, and I don't want to pay that much for water." What will we do? Go to war, invade Canada? They are going to go war in the Middle East over water. They have terrible shortage. I'm sure the folks in Iraq have figured out what to do if they're going to discover that they don't like Turkey's control over one of their water sources. So without an appreciation of history, people disregard relationships and obligations and laws of consequences.

In computer science, it's the same thing. Do you really think we would have invaded Iraq if we had remembered that people in that part of the world remember-- for example, in Persia, they still sing songs and tell stories to their children about the Devil incarnate, who happens to be the guy we call Alexander the Great. They have a long memory there. If we had known a little bit of history, would we really have invaded Iraq? I think not.

There are similar themes in computer science, but you've come to the end of the interview and it would take quite a while to show you the details of these histories and how expression evaluation got mangled when people decided they wanted a one-pass compiler that it would fit into 32k words. What that did forced compromises on the way you evaluate expressions in floating point. Now these rules are taken for granted as if they were built into the language and taught as if it's part of language theory but it's not, it's just an accident of history. So I feel that we should make way and make a place for history in the curriculum, but since you're the historian you're going to have to figure out a way to package it. What I would like you to do is help us figure out what we should take out in order to put history in.

HAIGH: Oh, probably the numerical analysis.

KAHAN: Well, there is no numerical analysis to take out now. That's been taken out already.

HAIGH: Maybe the artificial intelligence.

KAHAN: Well, that's a good point. It's a fad now, artificial intelligence and soft computing and multimedia and things like that. Who knows what will be in the curriculum later? It wasn't so long ago that we were teaching Pascal. This was the great structured language—and now where do we teach Pascal? It's gone. How long did people think Java is going to last? It's an abominable language. It has its good points, but it has a lot of bad ones. How long do they think that's going to last? If you have some historical perspective you may, for example, realize that if you plan to work in the computing industry for more than ten years, you're going to re-learn a lot of stuff. Are you prepared to re-learn that stuff? Are you prepared to negotiate working conditions so that you'll be able to spend some of your waking hours investing time in the advancement of your own capacities? A lot of people don't think of that.

Well, all right, what else?

[Tape 12, Side B]

HAIGH: So, unfortunately, the end is now nigh. But I will conclude by asking you the same questions that I have asked the other interviewees at the end of the interview. The first one of those would be, looking back over your career as a whole, what would you say that your biggest single regret is—either in terms of a mistake that you think you personally made, or just in terms of something where you wish the world as a whole had gone in one direction and, unfortunately, it went in another.

KAHAN: Well, I think I've mentioned a couple of my mistakes. I think the important thing about mistakes is that everyone is going to make them. If you don't make any mistakes that means that you aren't trying to do anything hard. One thing I've tried to do is not only to correct my mistakes but to make sure that the people around me understand that they also have the right to make mistakes as long as they don't just sit here. Fix them.

My biggest mistake was not mathematical. My biggest mistake was not spending more time with my sons. They're interesting people now; they were interesting then, but my wife acted as a kind of buffer. She saw that I was preoccupied with things that were concerned with my career, and so she sheltered me from the consequences of the inadequate amount of hours that I spent with my sons. It wasn't that I spent no time—I got them to help me fix the cars so that they would learn how to be handy about these things. We took them off to various places and we often went off into Tilden Park near here. So it wasn't that we spent no time and it wasn't as if I was an absentee father; I was absent more than I could have been and, had I spent more time with them, I would have enjoyed their company. My relations with my sons are not so bad now, and so it's not because I have to regret some schism in the family. That doesn't exist, so far as I know. It's rather I would have enjoyed their company more and I think that would have made for a better-rounded life. I envy the wisdom, for example, of Jerome Coonen, who decided that he wanted to spend time closer to his sons while they were growing up and I say that I wish I had been that smart.

But the best thing I ever did, which I've mentioned already, was of course to capture my wife, because practically everything good in my life has come about because of her. First I had to deserve her, and that meant that, instead of being a lazy slob, I had to become industrious; after a while that became a habit. Second, I had to be more considerate of people than would be my natural inclination, because otherwise I would have disappointed her and perhaps even embarrassed her.

HAIGH: Although you had said when the tape wasn't running you didn't have to worry about making the rest of the world love you because she did.

KAHAN: That's correct. But that's different from being inconsiderate. So I didn't have to seek other people's approval or admiration or affection. I get ample dose of that at home. But I think I would have been inclined to be oblivious to people—I've often said unto my wife that flesh heals, but metal doesn't. So I try to be considerate of devices. You look at my computer for example: it's several years old, but it's very clean. I maintain these old computers that keep on running; I maintain my old car, which is 22 years old; I fix things around the house that function as best they can, considering their ages. I would have been inclined to figure, don't worry about people; people can recover from various slights and insults—but maybe they don't. So I have to be more considerate and I have to be more patient. I think that the fact that I work hard instead of like a lazy slob, the fact that I am careful and so on, these are all habits that have accrued mainly because I felt I ought to deserve my wife.

Of course I look and say what good has that done? I have solved some pretty hard problems. Sometimes these are problems that wouldn't have mattered a hell of a lot. The problem I solved in my thesis about explaining why successive overrelaxation behaved the way it did for diffusion problems—one could say that is very well and so on, but we don't do that so much anymore. We do something else. When we do use overrelaxation, we use it under circumstances where your theory doesn't apply, so we still can't explain it.

I think my understanding of error analysis has helped me devise algorithms—for example the singular value decomposition algorithm that I devised with Gene Golub—that have made quite a dent. I could go on and on. There's the notes that I have on solving linear equations where one of the theorems says, if Newton's iteration converges, no matter where you start at some interval, to a place where a function actually reverses sign. Then the secant iteration will converge, no matter where you start it. There is an example of a theorem that is excruciatingly difficult to prove. The note you have here is "why tangents and secants will do". That's an old version of that paper, but the newer version is posted on my Web page. I suppose somebody could scoff at that and say that's not terribly important. How do you know when Newton's iteration converges? Actually, my paper gives you an indication of when Newton's iteration will converge and it's fairly general and some other people have discovering similar ideas. That's a very hard problem, so I can be proud of the fact that the difficulty of a problem doesn't deter me when I feel that, if I work at the thing hard enough, I will either understand why this all works or I hope I'll understand why we can't understand it.

Why can't we understand this problem of conserving confluence that curbs ill-condition, and therefore we'd like to understand this very intricate structure of self-intersections of what I call pejorative surfaces. Well, ultimately, we discovered in general that it is combinatorily daunting. The problem that you actually have is not necessarily the one that is *in general*. So it's still worth pursuing, even though you know that the general case may be intractable, but maybe your special case isn't. The stuff I'm doing on differential equations certainly interests me. Think of the numerical method that steps across poles—it just steps across them, and you get a result as if the pole weren't there. But it's the right result. Obviously there are some interesting things that I've done mathematically. I can't single out any single one as the thing of which I'm most proud, because what I'm most proud of in that respect is that I'm still persistent, unwilling to give up. Or as the old cigarette advertisement used to say, I'd rather fight than switch.

As for the mistakes, I've told you about the mistake of not following through on the 8087 stack that has had grievous consequences. I'm sure there are lots of mistakes that I've made some of which I may not know about because for all I know I've bumbled through life like Mr. Magoo, with very dim eyesight and hearing that's not all that good either. In fact, I've suffered from tinnitus for 30 years. In this conversation what you may not realize is that sometimes your consonants get lost because the tinnitus is by far the loudest noise in my head. I know it drives some people nuts, but I was already there so I didn't have far to go. But it's possible that I bumbled through life like Mr. Magoo had—causing, for all I know, consternation and even damage—and nobody tells me. What can I do? I'd like to think that a little bit of self-doubt is a good thing to apply from time to time. But on the other hand, if you were to ask somebody, "Does Kahan have self-doubt?" they would probably say, "No."

HAIGH: So it's not necessarily a piece of functionality that you are exposing to the outside world?

KAHAN: Well, it's just exposed a little bit to you and every now and then I'll tell my students this same advice: sometime a little bit of self doubt is worth applying. Just from time to time—not too much of it.

HAIGH: Well, that concludes all the topics that I've prepared for the interview. If there's anything else you'd like to say at this point then we can take a couple of more minutes.

KAHAN: Well, I look forward to what you're going to be able to make of all this, but I don't envy your task; you've got a lot more than you expected to get. It's more disorganized than you would have liked. And of course as a historian, you'd like to think that there is an organized way to see these things that helps you understand them. But of course you run the risk that life and history are entirely accidental, not random. Accidental is random. It’s not that there’s somebody rolling dice and the outcome of that determines life, but our ability to appear into the future is limited. Therefore occasionally we do things that have unintended consequences. That’s not random it’s just accidental.

The essence of a moral person is the consideration of consequences, of choices that could be made. And then the consequences of the consequences. Not just the consequences to oneself, but the consequences to others because if you disregard those there will then be consequences to yourself. And then the consequences of those consequences until you reach a natural horizon beyond which you cannot peer. If you have given thought like that to the consequences of the consequences of the consequences, out to this natural horizon, then you have an opportunity to make a moral choice. Why should you make a moral choice? Well, only because it’s harder. The easy things after a while they get to be not worth doing over and over again. The hard things seem to be the ones worth trying. So what you’ll have to do as a historian is wonder, I think, whether you can discern a pattern from which you can distill a principle or two or ten or a hundred, some finite number from which all this makes sense.

I don’t worry about those things. I’m concerned about history but one of the things I don’t worry about is whether or not there is some overriding principle or principles of history. I’m not sure that there have to be any. I think that a great deal of history is accidental. What we can learn from history best then, is that some things can happen, which if you were the writer of fiction you might never imagine. So history has as its principle value, I think, the stimulation of your imagination. A narrow life deprived of an acquaintance with history with biographies and so on, that is an imagination starved, starved for stimuli that history alone provides. So I hope you’re able to find a principle, but you might not be able to; in which case all you’re going to have to do is record a sequence of episodes and hope that they are entertaining.

HAIGH: Well, with that quote then I will thank you for an exceptionally thorough and thoughtful interview.