Oral History of David May

Interviewed by:
Kevin Krewell

Recorded: August 16, 2011
Mountain View, California

CHM Reference number: X6237.2012

Professor David May, August 16, 2011

Kevin Krewell: All right, well, we're here with Professor David May. Thank you very much for coming to our museum and spending some time with us.

David May: It's good to be here.

Krewell: Could you start by introducing yourself—your name, your title, and what do you do?

May: I'm David May. These days I'm a professor of computer science at Bristol University in the UK. But I spend half my time as the Chief Technology Officer of a company I started five or six years ago called XMOS.

Krewell: Can you give us a little background on where you're from and how it was like growing up for yourself?

May: Well, I was originally born in the UK in Yorkshire which is—sort of—towards the north end of England. And I stayed there until I went off to Cambridge University. I was at Cambridge 1969 to '72, I think, in what was then the mathematical lab and—although I started off doing mathematics—I then studied computer science in the final year. I moved on to Warwick University to do research in Robotics and then moved on to INMOS when it was founded in Bristol in 1979.

Krewell: So you started off in robotics and then wound up in computers.

May: Well, actually, I really started off in computers. I was fascinated by almost anything that moved from when I was age three or four. And I pulled things apart as some kids do and I put them together again and built other things with them. My father who was a teacher spotted this sort of aptitude. He wasn't a scientist at all. He was a language teacher, although he had some background. His father had been an engineer of some kind; I think in the ship building industry. And so he asked around—what do you do to encourage my talents in this kind of area? And a rather enthusiastic maths teacher suggested a binary adder, a binary calculator. And so they built one of these things for me. There were two rows of switches. You put binary numbers in and little lights at the top of it produced the output.

Krewell: What age was this?

May: This was about when I age 10 or 11, I think. But I had been doing electrical stuff since before that actually. And this really switched me on. And I suppose computers were becoming known about by that time and I guess they were the ultimate challenge in terms of things to get your head around—and the most complicated machines in existence, or about to become so. So then, armed with quite a lot of equipment—we found it easy to buy gadgetry on the streets those days. I mean they were throwing all of the equipment out post Second World War. And you could also buy useful stuff from the telephone switching industry, of course, relays, and things. And so I started building calculating machines when I was a teenager. I always knew from that point on that I wanted to be involved in building and using computers.

So how did I end up doing what I was doing? Well, there was no computer science program in the UK when I applied to university. I think there might have been one or two, but I was sort of headed off to Cambridge, or that's where people seemed to think I ought to be. So I went off to do maths. Fortunately, I've always been good enough at maths to get through the educational system on it, although my real talents are not mathematical. My real talents are building things. And so I went off and spent two years doing maths at Cambridge. And by that time, fortunately, they had created a final year program in computer science. I then studied computer science for one year and then graduated in Computer Science. Now, the people who taught me were great people. They were Maurice Wilkes and David Wheeler, who built the EDSAC 20 years earlier than that. And Martin Richards, who was the designer of the BCPL language that was the precursor to C, taught me all I knew at that time about programming languages and compilers and stuff.

I'm not totally sure where the fascination with Robotics came from, although actually I still rate Robotics and Artificial Intelligence as the things we're really all trying to do. I mean, computers are just a step on the way, I think. I think most of us are really interested in things that interact and things that move as well as things that think. And so I went off and started doing that at Warwick—a fairly young university at the time. And at the time a robot in a lab was a huge machine that walked up and down—or rather, trundled up and down—on wheels and had an umbilical connected to the machine at the corner of the room and pointed a camera and spent a long time thinking and said: oh yes, that's a teacup or whatever. There's only a choice of teacup or no teacup! This wasn't a very difficult job! But at the time it was quite challenging. Anyway, I thought this was not the right way to do robotics at all. I mean—I wanted my machines to be self-contained, no umbilical, all self-controlling and whatever. And around that time, of course, Intel started making microprocessors. And I took one look at these things and thought: the way to do all of this stuff is to dedicate processors to particular functions on the robot, the actuators, the sensors and so on. And then you network them all together and program them and you build the control system.

Krewell: So a distributed control system.

May: A distributed control system around a mobile robot.

Krewell: As opposed to minicomputers sitting in the corner with an umbilical that dragged along behind it.

May: Exactly. So this was my plan. And being young and enthusiastic at the time I thought I'd spend a few months just sorting out how to put together this kind of distributed control system, how to program it and so on and then I'd go back to the robotics. And I'm still hoping to get back there. <*laughter*>

Krewell: Well, you're right on the microprocessor side because they are distributed controllers and it's in your car.

May: Well, absolutely. So I mean it was bound to happen eventually. It's just that you have to be aware of this—at the time, this is 1971, '72, '73—the world was barely even doing distributed computing. So I had taken on a fairly big challenge to invent all of this lot from scratch. But it became very interesting, partly because distributed computing was just beginning to happen at the time. And I found myself amongst a very interesting group of people, in different research groups in the UK, who were looking at distributed processing from various different perspectives. And, of course, the same was happening over here in the States. And people were discussing how you program these things, what the operating systems were going to be, how the machines should be structured. Should they be message passing machines or shared memory? All of these core things were very much in the air in the seventies. And I came into contact through doing research in those areas with people like Tony Hoare—who was doing Communicating Sequential Processes—a programming language technology for concurrency and parallelism. I came into contact with the Iann Barron who was one of the founders of INMOS, who had previously run this computer company, Computer Technology Limited, developing Modular One minicomputers that I had used. And really there was quite an interesting network, quite a lot in the air at the time, sort of thinking about how to build distributed computing systems, parallel computing technology and how to develop the programming languages and tools to go along with them. And I think it was very much in that context that the UK side of INMOS came together.

Krewell: Tell us more about how INMOS came together from your perspective. You weren't part of INMOS at the time.

May: Well, INMOS is an interesting story all by itself. I mean INMOS was created in a context where—I mean—the UK was in bad shape. It had just largely failing state-held industries in coal and steel and whatever—and there was a lot of inflation. In desperate need of fairly serious re-generation. And at the time in the late seventies we had a Labour government and they were minded to start intervening with investments, in part, trying to rescue some parts of these existing industries, but also they were thinking about the possibility of investing heavily in creating new ones. And there had been some discussion about the idea of microelectronics and computing and so on. And they eventually created a thing called the National Enterprise Board, which was essentially a state-held venture bank. And so Iann Barron had been involved in advisory roles to government whilst they were thinking about all of this. Now, I don't know the details of the story here.

The American side of INMOS was created by Dick Petritz and Paul Schroeder. Dick Petritz—they'd just finished Mostek essentially and got out of that and Dick was looking for something else to do—[something] which he carried on doing throughout his life as you probably know. And at some point they had figured out that the only possible way to get the investment you needed for semiconductor manufacturing in a new company at that stage was going to be either a government or an oil company or IBM—because nobody else would have enough money. So they were obviously hunting around for one or other of those. And there was what appears historically to have been an almost chance meeting between Dick and Paul and Iann Barron at a meeting in the United States at a conference. I've always wondered how much of a chance this was. I mean historically it might be interesting to see whether it was actually set up carefully or not. But anyway, what happened was that these three came together and put together a proposition to create a new semiconductor company on the basis that the UK government would put the money in. It would get moving quickly in the United States by exploiting the expertise that they had access to in RAM manufacture which is what Mostek had been doing. And so they were going to set up in the States doing RAMs quickly which is an easy route to the market—because you make one that's better or cheaper than the competition and you can just steal the socket. But the longer-term proposition was to be in the UK and it was going to develop microprocessor technology. Which, of course, takes longer to develop but locks the customers in. Once you've got their loyalty they don't move away.

Krewell: So they were going to build fabs for the…

May: And the fabs were very important because this was about creating jobs and revitalizing areas of the UK which were in the doldrums because of the industries there failing. And so, as it happened, in 1978 they made the decision to go ahead with this and committed 50 million pounds, which was a lot of money at the time, to investing in microelectronics. And at the time the story historically is interesting because Iann had said to them: if you want to create an industry you've got to start somewhere and investing in start-ups is a good way to do it. And so INMOS was founded on that basis. And, of course, the question that had obviously gone through Iann's mind was: what are we going to actually build as a microprocessor? Are we going to make a better copy of somebody else's? Probably not. Are we going to do something new?

Krewell: So the idea was he wanted to build a memory business, but then he wanted something in addition to the memory business.

May: Absolutely—because the memory business is easy to get into but it's easy to get pushed out of. So strategically, you know, this was not going to be a good long-term plan. Whereas, if you use the memory business to get the company started and then put in something which has got the basis for a much more stable long-term business which locks lots of customers in once they've adopted your technology then— that was the plan, OK. And I think that was probably quite a good plan, actually. And so that's what happened. And it wasn't to me entirely surprising that the idea of the transputer, a word which Iann coined <laughs> . . .

Krewell: A combination of a transistor and computer.

May: Yeah. The concept was that the industry needed to move up a level of abstraction in chip design. And it had been stuck with essentially—first, transistors—then, in a sense, logic gates in the 7400 series type stuff is one step up. And then if you want to make another step up where do you go? And the idea of the computer essentially as a component from which you can build things as complex as you like, was what he came up with. But—as I said—I mean it wasn't purely Iann's invention, because actually prior to that I had been building things which were essentially—at the board level—microprocessors connected by communication links for doing applications like robot control. And other people, of course, had talked about similar things. But the idea of actually bringing them together into a single chip, bringing all of this stuff and making a single chip computer as a programmable component for electronics was, I think, probably Iann's originally, and he coined the word and wrote it into the plan.

And I think what happened then was that—Tony Hoare was, of course, well, known because of his contribution at the time, 1978, the CSP paper Communicating Sequential Processes—which had this notion of software processes communicating by message passing which is a very natural fit to an architecture which has got computers joined together by communication links. And I met Tony in about 1975-76 because I was, of course, doing this work from a rather practical point of view on building robot control systems. And he was working more on the language and language theory. And we got to know each other quite well. So he was drawn into by Iann as a consultant in 1978, and so was I. I got a call out of the blue one day, and so I had been in Bristol for about half-an-hour and he said, "Well, are you going to join us or are you going to be a consultant?" <laughs> So I thought about it for a bit and I thought actually this might be an interesting thing to do. So it's '79. I had a university contract so I couldn't just jump up at the end of '78 and leave.

Krewell: How long was your contract for at the time?

May: So I stayed——well it was a permanent contract.

Krewell: You had to take a leave**?**

May: I had to give several months notice because—as you do as an academic. So I actually formally joined the company in '79 but I was actually doing quite a bit of work for it—in the early thinking about what the actual shape of the was going to be—from about late '78 onwards.

Krewell: Who was the early team that started to put all of these concepts together that eventually became the transputer? How did that team kind of grow?

May: Well, this is interesting because, actually, there was a lot of coming and going at that time. I mean really, looking back at what information I have about this there were loads of meetings and memos and whatever which are discussing possible approaches to doing a transputer. Of course, at the time, we were only just getting going on the process technology. So there were also issues—we just didn't know what we could fit on a chip. I mean, remember, in 1978, '79 it was taking an entire chip just to put a processor down and we were trying to put a whole computer down with memory and communications and everything. So we were trying to figure out what we could actually potentially build. And then, of course, how to build it in such a way that it was going to be usable. And so the team was being built. At that very early stage, when I first started working there as a consultant, originally, there was myself; we had Jonathan Edwards who was our process technology guy. And he was interfacing quite a bit with people developing the process technology because we were developing process technology. That's what you did in those days. And one or two more people, more of the people who would be putting the company together—sort of personnel and management people and things. And gradually the team was brought in around that. There was a very interesting incident in the early part of 1979 when we decided that it would be great to bring one or two new graduates into the company. And we invited about—rather at a late stage we sent notices around a lot of good universities and got a lot of applications. And we eventually invited 21 people to Bristol as a sort of interview and introduction to the company and so on. And our people at the time said well, these guys have about four or five job offers in their pockets. So if you're thinking of hiring four or five of them you're going to need to make all of them an offer. So we did - and every single one of them accepted it!

Krewell: Instead of four or five you went with twenty.

May: By this time we've got about ten to a dozen experienced people in the company. But I mean I was one of the most experienced and I was, I think, 28 or 29 at the time. So it was a pretty young company. And all of a sudden we jumped by another 20 almost all raw graduates. And so my wife used to say: this is like the students' union in here. But it was a fantastic catch because these guys were—they had not much experience but they were very clever. Mostly in the UK they had a choice between a rather aging electronics industry, we had Plessey and GEC and nobody wanted to work for them if they were really smart. And so these guys were otherwise just going to do something completely different. And so we caught a lot of the best talent at the time and we carried on doing that over several subsequent years. Of course, the legacy of that has been that all of those people have gone off into doing all kinds of different things. And what Iann Barron and our politician Tony Benn who was involved in setting this thing up said about—if you want to start—want an industry—you've got to start somewhere—it's turned out to be true. It was a training ground for all these young people. Anyway, it created a very interesting and a dynamic culture inside the office. And really, things didn't start to stabilize until about 1981, as far as I can see. I mean certainly, my earliest notes that can be seen to have the beginnings of what the transputer actually became date from mid '81. And so the team I really think that we're talking about from the point of view who put it together is the team that was the most active people in about that time. And there were several of them because we were innovating in so many different places. I mean one of the interesting things about INMOS was that Iann Barron—the UK founder—basically thought that almost everything was wrong with computing. So we were going to do everything new. <*laughs*>

Krewell: It was a clean slate.

May: And so it was a clean slate. And so we were simultaneously designing a new architecture, a new instruction set, new ways of putting the components together, a new programming environment, a new language. And so it was almost the opposite to what you'd expect. Normally, you'd expect the grey-haired guy to actually sort of dampen down the enthusiasm from underneath but actually Iann was more or less igniting the flames.

Krewell: He was cranking up the volume. Yeah and a lot of companies get started by knocking off other chips or second sourcing to get into the business.

May: Yeah.

Krewell: Here you guys were saying no, screw all that.

May: No, screw all that.

Krewell: We're going to just do a whole new thing. We're going to do everything from the ground up and everything else that's come before it is wrong.

May: That's it.

Krewell: Well, so was Iann in complete control? Or did he still have a board, a government agency that he had to hustle up to?

May: Oh, no, there was the whole works. And it became terribly complicated because what happened immediately after this investment had gone in was that the government changed. And the government that had believed in state intervention of this kind changed into one that didn't. So then we had an incoming Conservative government who looked at this [National Enterprise Board] stuff and didn't know what had been done through it. And in fact wondered what on earth are we doing with all of this state investment in everything from the old industry to these new things? And, of course, they didn't want to do anything too dramatic especially as this thing was supposed to be creating serious new employment in parts of the country that were in bad shape. So politically they couldn't just close it down—so they had to stick with it. And Iann became very heavily involved in a lot of the politics of both interfacing with the UK government and trying to actually—as you would expect in a new start—deal with his own board and then the American side of the company. And I think he got pretty busy. So my observation was it might have been a good thing because it actually meant that there wasn't quite so much day-to-day involvement in the technical stuff.

Krewell: Because he was too busy off on…

May: He was too busy…

Krewell: …trying to keep all of the balls in the air so to speak.

May: Yes, that's right.

Krewell: So you said around '81 things you think started….

May: Well, I've got a trail of documents which I've hung on to which are—where you can see the thing going—the successive iterations of things like the instruction set appearing. And they are recognizable in the sense that the documents towards the end of '81 you can match up against what finally became the transputer instruction set. And you can see the core of it developing there. And then other things come in like the technology that we used for enabling us to put the memory technology on the chip. For example, nobody had ever put together a piece of dense SRAM or DRAM style memory alongside a processor on a chip before. And it sounds perfectly straightforward until you actually try and do it. And then you discover the processor is almost certainly going to start generating so much noise that it upsets all of the operation of the memory cells and the sense amplifiers and so on. So that was another piece of technology that had to be developed. And then there was the whole issue of how to deal with the clock generation and everything. We were the first company that built phase-locked loop clock generators to multiply low speed clocks. We were fairly obsessed about making this thing easy to use because—if the idea is to make a component that can be used as easily as a logic gate or something then it better be easy to use. You can't ask people to feed in four-phase non-overlapping clocks at huge speeds like what other people were doing at the time. The TMS-9900 had a very complex clock generator chip. And they had to supply a clock—a separate chip—to the processor in order to generate its clocks.

Krewell: That was very typical at the time. A lot of the processors had external clock generators in multiphase.

May: Yeah. So all you needed to do with ours was to stick a clock reference on the outside at five megahertz and then the rest was done by magic, or more specifically by a phase-locked loop that was put on the same chip.

Krewell: Well, at the time phase-lock loop was magic.

May: It was magic. So we used these for clock multipliers and stuff. So this team was pretty interesting because there was the digital design part of it. There was the architecture part, which was mainly me. And then we got the people who were skilled enough on the analogue design side to do the phase-locked loop clocks—and the communication links that, of course, all had to be resynchronized on entry to the clock domain. Then, again, there was no requirement for common clocks across these systems. You could just feed different clocks into all of the s—provided they're roughly the same speed it would all work properly. And it [the team] came together. And then we had to have this more specialized area which, of course, came partly from the U.S. side of the company, which was the RAM design. But even the RAM design had to be done in such a way that it could be sat next to the processor and the communication system without them all interfering with each other and upsetting [the RAM].

Krewell: OK. So you were mixing memory logic for processors and analog circuitry, Serdes [serializing-deserializing circuit], and phase-lock loops all on one die, which was really very uncommon.

May: Extraordinary.

Krewell: Extraordinary is a good word.

May: Yes. And then, of course, the other side of it all was programming tools, a programming language, a development environment. And along the way we had the challenge of dealing with the computer-aided design process, which was another one where we actually effectively invented our own. We were pretty astonished by discovering—the original plan, of course, was to take the same technology that was used in the U.S. and start and use it. And we bought one of these things these—I think it was a Data General machine with the tools on it—and realized we basically bought a drafting system. They were selling roughly the same technology to the architects for designing buildings. So this thing was taking hand-digitized diagrams that the engineers had made and was storing them and so on. But it wasn't doing design rule checking or circuit connectivity or any of those things. I mean that was done by drawing off check plots and crawling over them with colored pencils.

Krewell: So it's just an electronic version of the old tape systems.

May: Absolutely. We weren't too impressed with it. But how we came to do it, again, we decided to build our own. And, of course, we had a stroke of luck because one of those 21 people that we actually brought into the company said to us: "do you mind if I don't join you immediately because they're offered me a place at Caltech for a year to go and study there." And he was a graduate from Edinburgh, which in the UK had already done quite a bit of work on VLSI design. So Eric went off to Caltech to learn more from Carver Mead at the time that Mead and Conway were doing all of this stuff. And came back and we started trying to build a replacement CAD system based on sticks originally. And the sticks compiler, you know, you put all of the different color sticks [to represent the design] and the compiler compacts things [to produce the layout]. Well, that went on OK for a while and then they discovered that actually this was not going to work very well. But then they iterated it. And basically Eric over a period of a couple of years put together what became the design system that we used. I mean it wasn't purely him but he did most of the clever stuff on the geometry and the design rule checking and so on. And so that was another sort of fairly major piece of innovation at the time.

Krewell: How big was the team? How big was the …

May: From my recollection I think we had about 50 people working in Bristol at that point, of which probably 30 or so was engineering. And at least it's of that order. I've been scratching my head about this recently just trying to figure out exactly who it was and what they were doing.

Krewell: You had the 20 students and you had 10…

May: Yeah, a lot of them were students. They turned out to be extremely productive.

Krewell: And then your role at the time was in charge of the digital logic and…

May: Well, I was actually mainly in charge of the architecture and the programming language. I mean I essentially designed the instruction set and the Occam programming language alongside it. It wasn't called Occam initially. It went through a whole series of different names, INMOS system language...

Krewell: And then Occam was from Occam's Razor.

May: And Occam was from Occam's Razor. So this was one of these designs that are fairly unusual in the history of computing where the machine has basically been designed quite closely in conjunction with an idea about how it's going to be programmed. I mean, other examples, I guess, historically are things like the B5000, which was designed as an ALGOL machine. And the KDF9 which was the counterpart British machine which was also designed largely with thinking of how to execute ALGOL 60—with the stack mechanisms and stuff from block structured language execution. So this was based on an idea that we had to effectively develop programming tools that would allow us to tackle concurrency and parallelism because that was the whole concept. And the machine needed to be designed in such a way that it would run them very efficiently. So essentially this was designing a programming language in conjunction with the instruction set that was going to run it—an architecture that was going to run it. And I was actually drawing on quite a lot of experience I had developed over the previous few years in compiler technology. So over this period I was actually issuing iterations of the language design, iterations of the instruction set design. And I was prototyping compilers that would map one into the other. So they were all developing concurrently.

Krewell: So before RISC [reduced instruction set computer], you had a minimal instruction set. So it was a pretty small set of instructions.

May: In a sense it was a RISC, although, not in quite the same sense as the word is normally used. I mean the core of the instruction set was incredibly small. I mean I discovered at the time that it takes something of the region of 20 to 30 instructions to do a reasonable job of executing the entire normal sequential bit of a high level language. And so if you look at the transputer instruction set there's an identifiable core which basically does sort of integer arithmetic, address manipulation, jumping, calling, conditional execution. And then there's a block of instructions that are slightly more application specific. They add more data types and long arithmetic. And then there's another chunk of it with just all of the clever stuff, the process and thread and task scheduling mechanism and the interaction with the I/O [Input-Output] system because in the , you just execute an input or an output instruction and the data just went off down the communication link or off to another process.

Krewell: So it wasn't memory mapped, the I/Os, or you just hit a—well, tell me was it memory-mapped or port-mapped?

May: I mean the way the communication was done was that the—you could use any memory location as a communication channel effectively. And this location either held a null value to say there was nothing there waiting to communicate. Or it was set to point to the process workspace that was trying to communicate. So one of them [the processes] will get to a point where it was trying to input, for example. If there was no corresponding outputting process ready it would just plant its identity [workspace pointer] into this store location. And then subsequently another process would come along and do the output. And at that point the processor would transfer the data and reschedule the stalled inputting process. And then all that happened for doing off-chip communication was that certain of these addresses were recognized as special addresses. So instead of the processor performing the communication it would hand the job off to a DMA transfer unit. So the chip had this set of DMA units that would deal with the link communications, the processor itself and the memory, as I've said before. The processor had some fairly serious optimization in the area of being able to do very fast block copies because it was the thing that was doing message transfers for all of the internal communications. So it did things like using the minimum number of word transfers to do byte-aligned block transfers.

Krewell: It also had the ability—well, this is part of the original that you could boot off either the link or off of a ROM [Read Only Memory].

May: Yeah. It had boot from link. Absolutely. If you set it to boot from link it would just wait for the first message that turned up on a communication link, load it into memory and then start executing it. And that's precisely how these larger networks of s were booted. You had this sort of almost viral like program that would actually boot itself into the first transputer and then send replicas of itself to the adjacent ones.

Krewell: So that's why a whole grid could just—and that's part of the initial early designs too.

May: It was pretty early, yes. In fact, the very earliest designs were simpler than the one that was released. I mean the very earliest ones were doing—everything was done very much word-by-word. And subsequently on the ones that we finally did we actually saw the message passing much more in terms of strings of bytes that were taken out of memory and put back. So it was more DMA-like in the final version. But even in the original word-based stuff the notion of reading a program from a communication link was there. It's there in the early patents that were filed anyway.

Krewell: And then the initial parts were 16-bit parts.

May: Well, the instruction set was—I mentioned Martin Richards originally, the BCPL connection—because I actually borrowed quite a few ideas from that. BCPL was one of the earliest languages which actually had the front end of the compiler separated from the back end. And in between was a language called Ocode, which was a stack based compiler intermediate code that he'd used. And actually several of the ideas for the transputer instruction set come from that BCPL background effectively. So I knew how to build, how to get these sort of byte stream instruction sets designed. And the transputer one is not the same as the Ocode one but it's similar philosophically. And the other thing BCPL had was a neat idea where—there were essentially no data types in BCPL. It has a model of storage where address arithmetic works, so essentially you've got a memory that is a collection of words and consecutive addresses. But it actually doesn't define the word length. So different machines will have different word lengths. And I realized when I looked at this that if you're careful you can design an instruction set in such a way that you can have different word lengths without changing the instruction set design. So effectively the 16-bit and the 32-bits have exactly the same instruction set and all that changes is the width of the data path. So that's what I did. So effectively you find the micro-code ROMs and things in the 32- and the 16-bit were essentially the same. And, in fact, it's even possible—and we contemplated—building a 24-bit machine, which may sound very wacky these days. But at that point in time everything was 16-bit, apart from things that were 8-bit or 4-bit. So it wasn't at all clear in the early 1980s that there was any particular need to jump from 16 bits to 32 and it sounded rather extravagant for a small component that was going to be difficult to fit on the chip anyway. So that was partly why it was done. In fact, we ended up releasing the 16- and 32-bit machines almost at the same time. We felt a bit of pressure to go straight to 32 bits because Motorola, I think, was just starting to release its 32-bit 68000 version and we didn't want to be seen releasing a sort of flagship wonderful new processor that didn't look as good as the competition's. So it was an interesting decision, that one.

Krewell: What were the other chips you were looking at the time of this competition, the Intel 432?

May: Actually, the 432 amused us I have to say. We stared in disbelief when Intel released the 432—as many other people did. At one point we actually had the transputer, the chip, the picture of it actually sitting against the background of a 432 chip. And, of course, the transputer is down here in the corner against this great big thing. And the architecture of the 432 was just something else. I mean they had actually—I subsequently discovered through a friend who had been working at IBM at the time—he said, I looked at this thing when Intel told me about it and eventually I had an idea. I called up my friend [at Intel] and I said "You haven't by any chance just taken a lot of people out of Carnegie Mellon University have you?" And he said, "Yeah, actually, they're the people that did the 432." I said, "Yeah, I thought so." I mean because they had this extraordinary Capability architecture in it. I had friends in Cambridge who tried Capability machines and things and you just have a lot of extra baggage to carry to do almost anything. And sure enough Intel found out and eventually had to can it. I mean I suppose in a sense it was almost—it appeared as a bit of competition because they were talking about using this thing for multi-processors and whatever. And it was just on a different scale from what we were doing.

Krewell: So it's really kind of the antithesis of what you were doing.

May: Almost the antithesis, yes.

Krewell: When you were designing the chip or putting the chip architecture together, were there constraints in terms of die size. I mean you have very few amount of registers. Was that purposeful? Was that limited by die?

May: Exactly. It was one of the things—the whole point was—I mean I was struggling to figure out how we were going to end up putting a processor in what, in fact, was going to have to be a quarter of a chip. I mean nobody wanted to build chips more than about 70 or 80 square millimeters. Now, I mean our manufacturing people were pretty horrified when they saw the size of the original transputer die. They just said—it's not going to yield—because they were these RAM people who were used to making things that were not the smallest chips but they certainly weren't as big as that. So there was a lot of pressure to try and figure out fundamentally what we could get away with. So in the original draft of the instruction set it has two registers in its evaluation stack. And I added the third one because I found that the cost of that was marginal. And it had a big benefit in terms of instruction density. If you put a very shallow stack in then you end up having to keep flushing things from stack into memory all of the time. If you put a very deep one, of course, it's diminishing returns. So the three deep stack was actually done on the basis of a fair amount of evaluation, of looking at compilers and programs, going through them and just seeing what the difference was between two, three and four.

And then, of course, what else have we got? We got this mechanism for extending the operand length. That's one extra register that was the prefixing mechanism which I was very proud of when I invented it. It was the result of an extraordinary diversion. At one point somewhere in that 1980 period we tried to build a machine—as others have done occasionally before—which manipulated variable length data, you know, everything variable length. Actually, that was part of the original thinking of the PDP-11—all of this kind of auto-increment auto-decrement addressing was there because they were trying to build a machine which could deal with byte strings—everything was byte strings originally. And then, of course, you suddenly realize that you can't seriously deal with things like addresses as byte strings or the machine just becomes too clumsy and slow. So you need the notion of a word that's at least the size of an address. And then it turns out to be useful to have that anyway. So anyway I'd been around this whole loop of trying to design a machine that dealt with variable length strings. And I came up with what was effectively a representation of the data items in which a long item essentially went prefix, prefix, prefix, prefix, end. And eventually I threw the whole lot away because I thought—this is all a waste of space. And then I turned my attention back to something more conventional. And I thought—actually this will be a great way to represent instructions. From a compiler writer's point of view it's a dream because essentially you've got—your most optimal instruction operand is the four-bit one and that was used to get hold of all of the top end of the stack and all of the small numbers and so on. And if you want a bigger number you just prefix it—if a bigger one you prefix it again. And so—compiler writers have struggled in the past trying to deal with instruction sets that offered them six different lengths of immediate operand. So I got all of the simplicity of the RISC architectures but without getting myself into the problem of having a fixed 32-bit instruction which is just too clumsy. You just get very poor density if you do that, which is why in recent years they've all had to go off into 16-bit variants and so on.

Krewell: Well, yes, firstly for embedded space where they need the compact addressing.

May: Well, the compact instructions is always going to be an advantage because it affects how much program you can get really close to the processor and it affects the instruction bandwidth, [and] things. So yes, the whole thing was trying to figure out, you know, how we could get this really small processor without losing a lot of performance and actually we did pretty well. Because it was small we actually kept the clock speed quite high, which always compensated for the fact that we were probably executing slightly more instructions. So the basic execution mechanism was just what you would think of as a stack pointer although we called it a workspace pointer because it was the thing we used for pointing at all of these different processes that were being executed. The program counter, the operand register for assembling long operands, and the three deep evaluation stack. And then on top of that we added a small amount of additional state so that we could have a scheduling register for maintaining a list of waiting processes in memory. And every time the current process became non-runnable, the processor automatically pulled the next one off the list and carried on.

Krewell: Almost like multi tasking or threading.

May: It's multi-threading as you might call it now. I mean the machine was basically a multi-threaded machine which would handle an arbitrary number of threads—up to the amount of memory that was connected—in memory—just bringing them into this very small amount of registers. Of course, there were vaguely similar things had been around before. I had used the TMS 9900—the Texas chip—which had a small number of physical registers and it used—one of which was used as a work space pointer and—it was a fairly memory intensive processor because all of the instructions involved taking things in and out of memory. So there's various bits of thinking that went into the transputer design. But that's how it was all done. I mean the control structure of the machine—which was the other issue as to how to keep it really tight—was done with essentially an almost conventional style microcode. We built a microcode ROM, which was synthesized, largely. And, if you look at the structure of that, about a quarter of it is actually the basic instructions that you need to execute almost anything. I mean the loads, stores, adds, jumps and whatever. Another quarter of it is these more specialized instructions to handle more data types and long arithmetic, and so on. And then the remaining half of it is actually the micro coded scheduler which deals with process creation, communication, handling real time timers. We had time ordered lists and things so you could do real time programs that would be all handled by the hardware. And a moderate level of prioritization. We had one priority level and one de-prioritized level. And that took a long time to write—that part of the micro code ROM was a tour-de-force. So getting it there and validating that it was actually going to function correctly was difficult.

Krewell: So was most of the inspiration for all of these design things seem to be—and being inspired by a lot of different computers at the time, a lot of different ideas, but was there any potential customers involved in helping to find the .

May: We didn't have any customers.

Krewell: No one was even scanning for customers?

May: We only had customers for RAMs.

Krewell: So you were doing a cleaned up design. And then after design was done or the first chips then you were going to start bringing it out to market and trying to find customers for it.

May: Yeah. Absolutely. It was an extraordinary project.

Krewell: So it's interesting because I mean that's like you build it first and they will come.

May: Well, it is interesting but actually it's—there are lots of ways you can build computers. So if you were trying to address something to a particular market then whether or not you've got any customers you can go and ask customers who are in that market for other people's products. So that's the sort of fairly well understood process and it's been used many, many times. You know, choose a market and develop something for it. But there's another way of doing it which is to say, OK so we're designing this thing to be general purpose. And there's a notion of designing for generality rather than designing for specific purposes. And so you're then looking at understanding what the fundamentals of the computation are, you know, how fast will this thing do vector operations, for example? How fast will it just run general-purpose code? How fast can it do branches and stuff? How fast, in this case, can it schedule tasks? And you can measure those against potentially competitive products or potentially competitive approaches, and then you can, as you start to refine things a bit—you can actually—get your tools together, and you can start compiling programs and stuff, and check out that you actually are able to deliver this kind of generality, because of course, the danger with designing for generality, is that you end up with a product that's not good at anything. So that was pretty much the way it was done.

Krewell: Was there something you specifically benchmarked against, was it like a VAX or was it more an 8086, 68000?

May: Well, I mean, I was conscious of issues to do with—more general issues to do with—were we executing more or fewer instructions for particular tasks? I mean, it's these slightly more abstract measures that I was initially bothered with. And then, of course, you start trying to use benchmarks, because you know other people are going to use them. And yes, we tried most of the usual things, you know, Dhrystone and that kind of stuff. Dhrystone measures almost nothing, as you probably know.

Krewell: But there wasn't—yeah, there wasn't a lot of good benchmarks, and it was just—

May: Yeah, there were not many benchmarks. So, a lot of it, you relied on fairly synthetic stuff, I mean, things you write yourself to explore particular facets of the machine, and in particular to look for places where it might fall down. It's always been an interesting challenge, actually, how to measure computers and what to measure them with. I mean, Iann Barron used to have a classic remark that you should always optimize—you shouldn't optimize computers for good programs, because most programs are crap, so you should make sure you could execute bad programs. Something I've remembered a long time.

Krewell: So antithesis of what a lot people do, they try to tweak it to get certain little code segments to run really fast, but it's—really, it gets all the general problems—

May: And of course that can result in some bad mistakes. I mean, building machines that are good at very specific things, and not very good at general purpose stuff, is very dangerous unless you really know where your market is. And actually, the evidence from a lot of history of microprocessor technology is that the chips end up not in the market they're originally intended for, so—success stories are often because somebody recognized, that a relatively general purpose device can be used in a particular place, and then it all happened.

Krewell: So when was the point where it all started to come together, and you got, you know, your first working piece of silicon?

May: Well, the first interesting piece of silicon was actually not one that has ever been seen. It was a thing called the Simple 42, which was a prototype that we ran, on a previous generation of the instruction set, one that was sort of halfway between my very original draft, and what eventually emerged as the final version. So this was a machine which was, I think, a 16-bit design, and you had essentially a similar instruction set to the one we eventually used, although there was some quite significant differences. The process scheduling was less sophisticated than in the final version. And we ran that thing. That must have been run in 1982-3, I think. And that was successful. It was called the Simple 42—you probably remember "The Hitchhiker's Guide to the Galaxy?"

Krewell: Yeah, sure.

May: Forty-two, the answer to every question. There was a fair amount of culture in the company because of the youngsters, and so the sequence of numbers sort of goes, you know, 14, 15, 16, 17, 18, 19, 42! So we built this prototype. Now the plan, originally had been to get the real version ready and out in, not long after that, I mean, 1983 or something, but there were a number of issues. Part of these were, I think, to do with—they were probably running the American plant flat out, and didn't want to start putting something different through it, displacing RAMs that were actually funding the company. The financial climate, of course, had become extremely difficult, because inflation was running in the UK at 25 percent over this period, so the 50 million cash that was provided upfront, was devaluing very quickly. And, in fact, I think another investment was put in to deal with that, to some extent. So there was a lot of pressure to keep the American plant running flat out, and so, in the end, we decided to run the transputer chips up in Newport, in the UK, where we got the new fab running. And so there was a delay for the final device. In fact, at one point, we—there was also a product that was done in the US, which was called the 424, which was actually a dynamic RAM based chip. This was a version of the which had four kilobytes of dynamic RAM on chip, and we eventually decided not to do that, and we switched the thing to a static RAM for the first design, and dropped the memory to 2K, so the first commercial chips were the T-414, I think, run in Newport, in—was this late '84 or early '85, it's round about then, so the thing was delayed quite a bit by a mixture of internal problems to do with keeping the company running, essentially.

Krewell: So I mean, that's pretty much five years after you started this project.

May: Pretty much five years after the whole thing was started, and probably three or four years after we had really got the project rolling.

Krewell: Before that, were you—before you had real working silicon, did you have simulators to run your instruction set on?

May: Yes, yes. Yes, we built simulators, and—

Krewell: And everything was homegrown, you had to build your own simulators, you had to build your own design tools, your own compilers.

May: Absolutely. We built silicon. We built the synthesizer for the microcode ROMs, and there was an optimizer for them. We built all of the libraries that were then put together by computer programs to assemble the data paths, things, so yeah, everything was ground up.

Krewell: I'm sure also contributed to the delays in getting everything put together.

May: Yes, a bit, I think, although, we were pretty much ready to go, significantly earlier than we eventually could do.

Krewell: And there were no foundry services at the time, so you had your own fab, and that you had to—

May: You had to work your own fab. And I mean, it was very much—it was a curious process in a way, because it was almost a constant, sort of, debate about what the design rules even were, you know—because the RAM designers, in particular, always felt they could bend the rules a bit to make the cells a bit smaller. So the negotiation process that sort of established a set of design rules and things, often still ended up with people taking liberties with them.

Krewell: Well, sometimes that can cause manufacturing issues.

May: It certainly could, yes. <*laughs*> That was why there were a lot of discussions about it. But we had Jonathan, who was an excellent interface to the process technology people, and I think he had a good sense of when they were serious, when they said, you can't do that.

Krewell: And at that point in time, do you have a sort of recollection of how many people were doing software, how many people doing chip design, how many were doing architecture?

May: Well the architecture people—was very small. You don't need a lot of—in fact, the last thing you want is a lot of people doing architecture, so—<*laughs*>

Krewell: Or circuit implementation?

May: So there were, essentially, there was a small group doing architecture, like, I'm talking small here. There was myself, there was Roger Shepherd, who did a lot of the microcoding work, and a couple of other guys, and there were people who were, of course, were translating that into the microcode itself, and the sort of top level micro-architecture, and then there were other people who were building these libraries and putting the data paths together, and it all had to be done, essentially, by hand, or by what we'd now call scripts. I mean, we didn't have scripting languages, you were just writing stuff in, probably a lot of it was written in BCPL, as a matter of fact, because we're in the era slightly before there had been any predominance between things like Pascal and C, and so on. And BCPL was the language that we knew well, and I had a lot of expertise from porting it around other machines in the past. So we built most of the computer aided design system on the—it was written in BCPL. And the machines on which it was run, were built by us as well. I didn't mention that, because this was before Sun workstations and things, so we wanted to have these—we built these things with big screens, and 68000 based processors underneath them, and built all the software for that lot, ground up as well.

Krewell: Even though 68000 was one of your target competitors, you were using it as a tool to build the—

Krewell: Absolutely. Absolutely. <*laughs*>

Krewell: That was probably the best microprocessor architecture at the time, the 68000.

May: Well it was the early—they had the 32 bit thing out, so this was important, because if you tried to run reasonably large chunks of software, then that's what you need.

Krewell: And then, once you started putting together—you got your first chips back, I guess the T-414, the first one? First commercially available.

May: Yeah.

Krewell: And then what was the next step? You now had to evangelize this language and chip to the marketplace.

May: Yes, absolutely, we did. <*laughter*>

Krewell: How was that, did you—

May: Difficult, difficult. And so essentially, it went like this. I mean, there was a lot of—it wasn't difficult to stir up a lot of interest in the UK, because this was a very conspicuous kind of thing to happen. The government put this amount of money into a high tech new chip company, doing stuff that was clearly leading edge, and drawing—

Krewell: At this point in time, the government still owned—

May: Well the government—yeah, the government still owned it, and so we were—and then the question is, how do you—I mean, the UK isn't a big enough market for this thing, so you've got to spread the word. And we tried doing that through a number of routes, I mean, conferences, and talking to people, and press releases and all the usual things. We actually did a lot in the Far East, in Japan, in particular. The Japanese had started this Fifth Generation Program, as you may remember, which created a lot of stir, and we got them very interested, and one of the very first presentations about the , and Occam, was done at one of their early Fifth Generation conferences. And that was very effective, because it simultaneously got the Japanese interested, and—but it also got a lot of people in the US interested, because they turned up at the Japanese Fifth Generation Conference, and wondered what this thing was about, and why were the Japanese taking so much interest in it. And I think that sort of caused people to take an interest, because it's never that easy to get people in the US interested in stuff that comes from over in Europe—at least that's been my experience. And actually, it's quite often difficult to get people in the UK interested in stuff in the UK, if they don't—

Krewell: Well that has changed.

May: It has changed.

Krewell: Acorn which led to ARM. ARM has certainly changed that now.

May: And I think things have changed now, but a friend of mine used to joke that there was a—when you were trying to sell UK technology in the 1980s—in the US, there was a, Not Invented Here, problem, and if you tried to sell it into the UK, there was an, Invented Here, problem, because all our people wanted to buy American computers! But we did it, you know—and of course, we also started building these prototyping kits. We had—and then we had the idea of creating a standard format for these things that were plus certain amounts of memory, so you could plug them all together, the Trams is what they were called.

Krewell: Right, the Trams, the modules.

May: And that was a great—that was almost a, by accident idea, that. We did it because of making it easier to assemble our boards for people to evaluate with, but of course what we actually did, was to open up a market for people selling Trams, and so we ended up with quite a lot of smaller companies that were sort of board-making companies, actually launching products that were more specialized versions of Trams that could do graphics and DSP and so on. And so that was quite successful, and that technology found its way into all kinds of places, I mean, not perfect, because a lot of it was research and academic, but then, you know, the question is, where else do you go with something new, because initially you're going to end up wanting people like that to pick the technology up and demonstrate that it's usable before it's likely that people would, in big companies, will pick it up to build product with. So I don't think we got that far wrong, in retrospect. I mean, one thing that was clearly wrong was the whole time scale. I mean, to go from zero with totally new idea, to big business, is a ten year project, not a five year project.

Krewell: Yeah, so what were the early successes, who were the customers that started the first ones to start adopting it? Do you remember?

May: I remember—the trouble I have is that I'm hazy on the dates of some of these. I remember some pretty interesting projects. I remember quite a lot of the ones in the academic world, because they picked it up quite quickly, and you started finding the thing in, particularly in areas like graphics and vision and image processing, and so on.

Krewell: And DSP. It also worked—

May: And all the DSP. I mean, we had some very early designs in radar, kind of large scale DSP, because radar systems, they were always having to configure them, so our technology was a dream for these people, because they—no two airports were the same, so they were always wanting a sort of parts kit for building a radar installation, and they could do it with our stuff—they just plugged the boards together, and assembled whatever they needed. I remember a very interesting—and then, of course, we got picked up by the, what was almost the supercomputing people, because they were busy trying to go heavily into parallel computing, and of course we had an early, very early spin out from INMOS, which was Meiko, which set off, and we placed an initial contract with them to build us a demo machine for showing off the transputer used in quite a large number. That machine was used to show the transputer doing fast, well actually, doing both fast database access, and delivering that stuff on screen in real time. And the example that was used, was the design database, so essentially, they did a fly-through demo where it was as if the whole metal layer and poly layers, and stuff, had been blown up to enormous scale, and you could fly down into the ALU and so on. It was very neat.

Krewell: So it was like an early 3D rendering, or—

May: Absolutely, so this was 3D rendering graphics and—but also, you see, we were great for providing multiple channels to disks, so early, sort of, Raid disk systems and things, you could easily build with s, because you just ended up with—we had the specialized transputer, the disk controller, M-212, which had a disk interface integrated together with the processor itself. And so you could basically just stack the disks up and then program the s to do all the distribution of data, and access to it. So Meiko built a number of these machines. In fact, one of them was used for things like, for doing real time election results at one point, in the BBC, and so graphics definitely took off in the more general purpose area, and we must have put together dozens of demos. In fact, I had a group of guys who became very absorbed in graphics, and I used to use them basically for—and let them go to demonstrate the technology, and we used to regularly put stuff in Siggraph. In fact, there was a box of tricks in Siggraph that used to run Mandelbrot sets and things, at the time when—

Krewell: Real time Mandelbrots?

May: They ran Mandelbrot sets in real time. You could literally take a window and go, click, and it would just go, bang, and it was there—this thing was built—this was an interesting accident. At one point in the early part of the manufacturing [of] these things, we ended up with an incorrectly programmed packaging machine. And so fully functioning die were coming off the back end of the line, were being inserted into the packages 90 degrees rotated, and then they were going on to the next stage and being rejected at post-assembly test. And by the time it was noticed that every single one of these chips had been rejected, 2,000 chips had gone through the line. So we, unfortunately, had to write the whole lot off, and I thought, what am I going to do with these things, and I had these boards made, that had 42 s on, and that was what would fit, seven by six on a VME board, and they were just absolutely wall to wall, and OK, I said, what are you going to do with them, but Mandelbrot is a perfect application. So in a little thing, it must have been a cubic foot, in the middle of the Siggraph stand, were 400 processors in a box, and so you could plug it into a screen, and you walk up to it, and there was the Mandelbrot set, and you could zoom into a thing and, straight away, there it was. And there was apparently, at one point, a man from, I think, Hitachi, came and looked at this box, and started looking around the back of the stand, to try and figure out where the supercomputer was. He thought this must be just a terminal. And then, having realized there wasn't one, he came and asked whether this was a new storage technology, in which we'd pre-computed the whole set of some enormous depth and stuff. But the one thing he couldn't get his head around was that this was actually a box of real time stuff. So anyway, that was a wacky application, but of course, that application is actually quite close to ray tracing, and ray tracing, there was quite a lot of stuff on ray tracing, and other forms of 3D rendering. In fact, the original multiprocessor Pixar rendering engine, and Renderman software, was put together on a box full of s, a little bit later than that.

Krewell: Anything that can parallelize in ray tracing and Mandelbrot sets, are all—

May: They're all parallelizable, I mean, my claim is that most things are parallelizable. <*laughs*> I mean, there are some known exceptions. There are some—there are some algorithms where they're very difficult to parallelize, although in fact, even with those, often you will find that there's an equivalent, or a completely different parallel algorithm that will do the job just as well or better, so—

Krewell: So going—actually, parallelism is—was kind of the important part of Occam, was the fact that you were dealing with the idea of parallelism in the language.

May: Yes.

Krewell: And no other language was really dealing with it in quite the same way.

May: No, the—no, and people tend to—the way I look at it historically, is that we got off— computing got off on the idea of sequential programs. I mean, if you think about it —a computer, in the 1930s, was somebody who sat in a room with a desk calculator, with a set of instructions supplied by the mathematician or the scientist for the calculation. So there's the computer. The program and the data comes along, you crank the handle and produce the results and give it back again. And in fact, the earliest computers, things like the EDSAC and Univac and so on, are all the same model. You know, the thing reads its tapes or cards or whatever, and does the job, and sends the results back. And everything else seems to have been bolted on since then. So you then decide it needs to interact, so you have, sort of interrupts or something, and then that's done by machine code programming, or some bit of an operating system, rather than by being done through the programming language. And again, this was something that seemed to me to be fundamentally wrong, when I was doing the robotics stuff originally. You want a programming language that can express the potentially concurrent tasks, and the message passing, or interaction between them. And I built, along with [my distributed processing system], again, pre-INMOS, actually I had a language called EPL—it was a message-passing language that I developed for this robotics work. And so—Tony Hoare had thought about this in terms of, from the perspective he'd come from—and so we were—when we started doing the system programming language for the , then the idea was to build a language which had parallelism and message passing and so on, built in. And last, but not least, the idea of, I call it event-handling and mutual exclusion, things which in the CSP, and in the Occam form, was modeled on the guarded command language. So you got this idea of being able to enable a number of potential sources of messages and then take the first one that occurs, and carry on, which is, essentially, the language based embedding of what otherwise has to be done with interrupts and device drivers and operating systems. And I'm a firm believer that this should be done in the programming language. There are then, just a lot of optimizations that you can do on it.

Krewell: Although, typically, the processors at the time would do that in hardware. There would be a hardware interrupt controller that would handle all these things, and—

May: Well, but then it still has to generate an interrupt on the processor, and the state has to be saved, and this is all done explicitly, and it takes a lot of time. And in the case of the , it was either just, it would be a small number of microinstructions, so your real time—the other way of looking at it is that the real time kernel is embedded in the hardware. And so it's a lot faster. <*laughs*>

Krewell: Well yeah, and that real time kernel allows you to handle these things, and not—quickly, and with a reasonable amount of time for each event. But that was tightly coupled to the language itself?

May: Yeah, yeah, the language I was designing with—from two perspectives. One of them was, I was making sure through building prototype compilers for it, that it would sit neatly on top of the instruction set, and I mean, there was literally a three-part document, which has each high level language construct, the method of compiling it into the instruction set, and the corresponding instruction sequence. So you can—and I kept iterating these things to get it all to line up neatly. Then there's a number of other—it dragged me into a number of—a lot of issues of language design that I had never quite expected to get involved with. I mean, I discovered that the—I must have inherited a lot from Iann Barron by this time, because as soon as I—because my original idea was that I would take a lot of existing ideas from existing programming languages and then reuse them, but then I discovered that, actually, I didn't like lots of what was in existing programming languages, and I probably invented slightly too much here, really, but it was a very interesting exercise. Because Occam has many things beyond the straight concurrency. I mean, it also has a very robust method of containing errors. I mean things people forget about, what happens when something goes wrong in a live system? And particularly if you're dealing with embedded applications, and real time control and robotics and so on, it actually matters that you think carefully about what happens when something fails. Otherwise you lose control of a mechanical system, and something undesirable happens. And so there were notions, the whole thing was actually—the transputer had quite rigorous error containment. The processor detected errors on things like arithmetic overflows, and other kinds of failure, and it caused the process to stop, and stop dead, that is. And this didn't prevent other processes continuing to execute, but it would stop the one that had failed, and thereby prevent data propagating outwards and errors spreading. So there are various notions in Occam that go beyond purely the concurrency, but did have an eye on the kind of application areas that I thought we were going to be in, and certainly, you ask about applications, but certainly we ended up with quite a lot of projects in robotics, and in fact, even three or four years ago— I had a project that I got into, which was to do with controlling these robots that are used for pointing TV cameras—well movie cameras, actually— so when you see these carefully programmed takes, they're done by big cameras on big robots, and they wanted to change the user interface to this robot, and I got a call out of the blue, to say that: “we've looked at the control system for this thing, and there's a thing called an IMS-T424 in it.” I said, "Oh really?" And of course, it was classic, and we saw quite a few of these. It was simply dedicating, as exactly in my original concept of the early '70s, dedicating a particular processor to a particular degree of freedom, and then networking them all together to get the overall coordination. And that turned out to be a very good model. So—and people used to say to me, you know, this thing is quite good for robotics. And I'd say, yes, there could be a reason for that.

Krewell: If you were—yes, maybe that was baked in.

May: But I mean, it's generic to real time control, and people evaluated or used this device for quite a number of control applications.

Krewell: Well considering how advanced, and how much thought you put into Occam, why do you think it hadn't caught on? Is Occam too tightly coupled to the transputer, that you couldn't port it to another platform?

May: No, you can actually run it—you need a small runtime kernel to replicate the effect of the microcoded instructions, but you still get a very fast and effective multiprocess mechanism.

Krewell: So had you thought of migrating it to other platforms?

May: We actually had other implementations of it running on other computers. I think—and certainly that was an original intention. I think it was one of those things that rather fell by the wayside for a number of other reasons, I mean, because the problem is, that the company ran into a number of problems, which is one of the things that plagued the latter stages of the 's evolution. I mean, it was, in 19—was it 85, or 86, the company was sold to—the government finally decided it could do a deal! Sold—

Krewell: it wanted to get out of the business—

May: Sold the company to Thorn EMI, and that was probably a serious mistake on their [Thorn's] part. Thorn EMI was a very strange conglomerate in the UK, which made everything from military stuff to light bulbs, and consumer appliances and was a record-label as well. So their chairman changed, and they wanted to get into something to focus much more on high tech stuff, and he went and bought the whole of INMOS, which, for somebody who didn't really understand what chip manufacture was about, was probably a bad mistake. And it turned out to be a pretty bad mistake, because—then the DRAM business collapsed, you know, there was a worldwide recession at that time, and so the main source of cash disappeared, and in fact, at one point, the main source of cash, on a month by month basis, was the thing that we had designed as a color look-up table that was going to be part of a graphics controller, and IBM put it in all the PS-2s. So we were churning these things out in millions. So that was a problem, and, of course, eventually the company was taken over again, by ST Microelectronics, and so there was a lot of issues there to do with really putting enough momentum behind all these things. These are all marketing and promotional things, really, and spreading the programming language across many platforms would have been a great idea. And it was actually part of the original strategy to establish the programming language first, which was definitely not a good idea, because it's very difficult to sell programming languages! But the general idea was that the programming language is not supposed to be proprietary, the programming language is supposed to be what the world needs, because of having to program concurrency, and we just happen to have a really good implementation of it. That was the thinking.

Krewell: But we never got to that phase. It—

May: Well we never—we still seem to be struggling to convince the world that it really needs to do concurrency. I'm slightly baffled by that after all these years.

Krewell: I'm actually baffled by that, too, because it seems to be the hottest topic in computer science these days, is which language to use for <*inaudible*> proprietary languages, and extensions to C.

May: Well, there's a pile of history behind that, of course, because the fundamental blunder, in my view, is that coming—having—we had a great time living in a simple world, where there was a single processor and an infinitely extendable memory. I mean, this is the Turing-Von Neumann concept, and it's great. You need to do something more complicated, you make the processor faster, or you extend the memory. These are the two degrees of freedom you have. OK, now when that seems to be running out, the question is, where do you go next? Now my take on it is that you replicate the whole thing and join the things up with communication links, and this takes you straight into, exactly, the CSP, the Occam and the transputer style model, OK? The other take on it is you cluster more and more processors around this single piece of memory, and this just seems to be never going to work. It just isn't going to work, but that is—seems to be what people think of as the easy way to do it. And superficially, it appears the easy way to do it, but actually, when you get into the details, it's not. Number one, shared memory programming is actually very difficult, so you're actually offering the programmer a hard time, which is probably why there are 101 ways to try and make it easier with one language or another, and number two, it doesn't scale at all well. It's, you know—essentially one processor these days can completely absorb the bandwidth of the big piece of memory. So then you have to go into a whole pile of caching schemes which make it even more confusing to people. So anyway, so I don't know why we are where we are.

Krewell: Well there have been attempts at—

May: I keep hoping.

Krewell: Well there have been attempts at NUMA [non-uniform memory architecture] <*inaudible*> architectures, and distributed memory. I mean, the later alpha processors, the—actually, AMD Opteron signs, they have local memory to each processor that then connect to—in fact, if you look at what is the Opteron, or, AKA, the Hammer Architecture, but it kind of looks like a grownup version of the transputer.

May: Well, good, I hope they get there. I mean, but it's also cultural, you see. Thing is that you take people into computing and you teach them programming, and the first time you teach them—the first thing you teach them is that the nature of the job is to break things down into a sequence of steps. Well, then you try and un-teach them. Because essentially, if you taught them that, and you then say, OK, the computer has to do two things at once, they'll construct you a program that says, well we'll do first a bit of this one, and then a bit of this, and then a bit of that, and then—so they essentially—the mental model is already into serializing things. And then, event handling is another part that I mentioned to all this thing, which people should be able to do as well, because most systems these days are interactive. So that should be part of your mental machinery. So the Occam thing really, I think, contains the three ingredients that you should teach everybody from, in CS-101, as it were. You know, it's concurrency or parallelism, it's sequential stuff, and it's event handling. And effectively, Occam provided a way of putting these things together tidily in a single language, and the transputer provided a good way of implementing them. And if you start to think about the programming task with those three ideas, then you see the world in a different way. I mean, essentially, you will look at a problem and immediately say, OK, there are five things we can do in parallel here, this is the way they communicate and so on. If you come at it the other way, and think, I'm going to write a single program that does all these things as a sequence of steps, and how do I then break it up? It becomes—it's just the wrong way to go. Because the breaking up is then going to be seen as a huge problem, whereas, actually, in many cases it's already broken up. A robot with ten degrees of freedom, it's already broken up—there are ten tasks to perform.

Krewell: So you're still teaching today, and do you teach those same—those concepts in your classes?

May: Well, I mean, I do. The only thing is, I keep trying to persuade my colleagues that this stuff should be in first year. We're getting there gradually. We've got it into second year now. But they all see it as an advanced topic, you see. And I just don't understand this, because it seems to me that if I've got—if you want to say, now do two things at once, that's what I want to say. That's all I want to say. I don't say, I don't want to say, now call the threads library to create a thread to [—and so on]—I just want to say, do two things at once.

Krewell: [How are we doing? Do you want to take a little bit of a break, or are you still keeping up? OK, all right, we're on a roll.] The first part, when you started producing this, how big were those? I mean, what was the limit of the manufacturing.

May: They were about—my recollection is that these were round about 60, 70 square millimeters, or maybe 80, I mean, we never went above a square centimeter. The T-800 was a bit bigger, because that had a floating-point unit in, although I think that was shrunk fairly quickly to a slightly finer geometry. So yeah, we would be willing to—there was—that original T-424 chip, with the dynamic memory, is instantly recognizable from the pictures of it, because the corners are cut off. It's a sort of slightly octagonal chip. And the reason was that, when they tried to put these things down on the reticle for the four inch wafer, the corners fell off the edge. So my endlessly inventive guys said, well we don't need the corners, there's nothing there. So we'll just—so in a four inch wafer, you know— you've got to be very careful about how big the chip area is, otherwise you lose a lot of yield, because things are just falling off across the—

Krewell: Because you're building square die in a round wafer, and <*inaudible*> a round wafer that fits in a round—And then, so now, looking back at—one of the things about the program was, it had so many innovations in one package, in one program. And we do talk a lot about Occam, which is one you're close to. It had on chip serdes which allowed you to chip to chip communication, which was, you know, almost glueless, I assume, there was, maybe buffers—

May: It was glueless, you could buffer it.

Krewell: For a standard length?

May: You could buffer it. It was remarkably effective if you—people did buffer it using fairly standard drivers and stuff, and it would run over meters. In fact, another product we made was this link adapter, because we found people wanting to connect onto—peripheral devices onto our links. And so I thought, well actually, the easy thing—it's easier for us to do this, so I got my guy to design a link adaptor which was basically a link in one side, and a parallel interface on the other, and we sold quite a lot of those, because they were better than UARTs [universal asynchronous receiver/transmitter]. They were—our manufacturing people hated me, because essentially this thing was the opposite of what they were used to. This chip was tiny, so this was 1,100 die per wafer, and so they said, "We can't handle stuff like this," and I said, "Well what's the problem?" You know, they said, "Well we're just not set up to deal with large numbers of tiny chips." So, but they did anyway.

Krewell: Yeah, and actually AMD made a part similar to that, called Taxi, which was a parallel in, and it serialized out, and you connected together and you could just—it was interesting, a very similar concept, late '80s, it was at that point, in bipolar <*inaudible*> But so that linked general purpose peripherals, and things to—

May: Yes, well we had two versions, actually. We had one that had a—was designed so you would easily connect it onto a bus, a conventional bus, because that was what we used for connecting s onto the PC development kits, development systems and stuff, and another one which just produced a handshake and eight bit ports, both directions.

Krewell: I'm surprised that, that was actually a very useful solution for many things.

May: It's very useful, yes, yes.

Krewell: And then and so that, you had the serdes on chip, you had the memory controller on chip.

May: Yes, the memory controller was another interesting thing. I've been trying to remember exactly who did this, because the was very good as an easy-to-use component, and the memory controller was part of that story, because we wanted to get, obviously a very good interface between the on-chip and off-chip memory, but we also wanted to take all the glue out. And so you know, quite a lot of effort went into looking at what kinds of memory chips there were on the market at the time, and what we would need to do to make a programmable interface, and so that thing was essentially configurable to mate with almost any dynamic or static RAM that was available, with minimal external logic, which is why all these Trams were so easy to make. They were—you look at the Trams and there's nothing on them, but the chip itself and a set of memory chips.

Krewell: Yeah, and the cleanness of the designs, the board designs is just amazing, because you see very little—

May: That's because someone thought about it.

Krewell: Almost everybody else in the business was having all these little peripheral chips to glue things together.

May: And I've watched this kind of thing happen, and it's actually—I've come to realize this from—even from my new company, that it's very easy, unless somebody really makes it clear that we want an easy to use component, the technologists will go and design a pad-ring [on the chip] that makes it easy for them[selves], without thinking about what's going to happen outside of it, and so then you end up bonding the thing out, and then you end up with all kinds of things that are, you know, signals in the wrong order which costs an extra layer of circuit board or something. But the transputer was done remarkably well from this point of view. They had got all that stuff sorted out right, and enough flexibility. I think partly, we must have taken on a number of people, in fact, the—I think the guy that probably did a lot of this, had been previously involved in Iann's previous company, doing a lot of board level design. So if you have a board level designer on the job of designing the external interface to the chip, then you will get the right kind of person to worry about it—

Krewell: That's true, because too many people will just lay it out for the convenience of the die, and—

May: You know, when you're designing, at the early stages of designing a chip, you don't think about hiring a board designer. You think of that as somebody you may need later to put the chip on a board. <*laughs*>

Krewell: And then so you have the on chip memory controller, you have a large amount of—a fairly large amount of on chip RAM, 2K, and eventually—

May: For the time, so—

Krewell: Yeah, today, cache is very—but it wasn't—it was just standard memory, it was memory.

May: It was just memory.

Krewell: It wasn't a cache, per se, right.

May: No, no, no.

Krewell: Did you have a memory management unit, or was that all done in software?

May: No, there was no memory management. The way the thing worked was that the compiler would actually—made it possible to allocate stuff between the on chip and off chip, and typically it would—and you could guide this to some extent with explicit configuration statements in the program.

Krewell: So what do you think were the best, the most successful part of the transputer program, for you, from your perspective?

May: Oh there's a good question. <*laughs*> It was a lot of fun.

Krewell: Well I guess that—it was a lot of ground up, new engineering that you could do as a fresh clean slate. You didn't have the—

May: Absolutely. Absolutely.

Krewell: A lot of unconstrained issues.

May: Yeah, I mean, I still basically believe that we proved a way of building systems, which I think will come back again. I will be very surprised if we aren't forced into that way of building things eventually. I mean, it will be slightly different in detail, but I just don't believe that you can build systems the way they're currently being built, and if you actually want to scale things as people do, to very large numbers of processors and things, then you will have to accept, full on, that these are concurrent systems, and you have to program them as concurrent systems, in some way or another. And I think the transputer idea proved that you can do a pretty good job of that. It's really 101 things. I mean, over the period, the number of innovations was extraordinary, and a lot of them keep coming back to haunt me, because, in a sense, because my patents keep being used as prior art to other things, so I get calls from lawyers who say, yeah, we've got this patent that claims to have, you know, done something with phase-locked loops or something. We think you did this a long time ago, or whatever, there's a whole load of those. <*laughs*> But you know, we built, essentially, the first computer on a chip. And the first thing that could be used directly as a system component, and I still think the computer with its links is a great way to build electronics. It actually, effectively gives you a standard component that you can optimize to death, ultimately, and get the real benefits of large scale manufacturing, and then customize everything by software, which seems to me, to be the way the world needs to move right now. On top of that, there are potentially interesting benefits that I realized, toward the later stage of this program. A component like this, managing its collection of concurrent tasks, threads or whatever you want to call them, actually can easily recognize situations in which it's got nothing to do, it's just waiting for more work, in which case you just switch it off. And building a large machine, or even a small machine, as a collection of devices like that, that are programmed to, in a completely natural way— I mean, when I need you to do something, I send a message, and you wake up and maybe you wake some more people up by sending a message to them, and then the result comes back to me, seems completely natural. It also is the way the brain works. So actually, a lot of it is asleep most of the time, and it just keeps—bits of it come to life and deliver the results—and go to sleep again. So now there's a huge concern about how we do energy management in computer system—both from the point of view of the mobile device, or the scavenging device, or the supercomputer, and there's your answer. You implement the thing as a large collection of small computers that only come alive when you ask them to do something. So that's what I think we did, we just did it a long time ago.

Krewell: And then what do you think you did—what could you have done better? What were the—what do you think were the—some of the, maybe problems—

May: Well, there were some obvious things that were probably, problematic from the point of view of the strategy. I mean, this was a big chip, not surprisingly, at the time. I mean, it was quite state-of-the-art, the yields weren't ever going to be brilliant, with a chip like that, and so trying to sell that thing as a system component, to be used, sort of almost like confetti, use as many as you like, this isn't really going to work, except in the relatively high-end embedded systems. And, of course, that's exactly what happened. It was very—it was very much liked in high performance computing, in high end embedded systems, robot control, vision, graphics, and so on, which were all, you know, the places where people were actually spending quite a lot of money at the time. But of course that high end stuff is never enough to sustain, in chip manufacture, in chip design, so that, I think, was a piece of the strategy that really wasn't thought out properly, and I'm not quite sure what you could do about that. Because these days, the world is quite different. I mean, the transputer, well it would be too small to implement, I mean, if you put the transputer down, you'd have to put 100 of them on a chip, before it became—

Krewell: There's this company called Tilera that's doing about 100 little—

May: So Tilera are putting quite a lot down on a chip. They're still rather bogged down in the cache coherent shared memory, I remember.

Krewell: But they could put a lot of die, and—

May: You can see the point. They, and Intel and others are demonstrating you can put a lot of die down on a manufacturable chip. So—the other thing, of course, was that the problem was that going from INMOS' point of view, going at the embedded marketplace—the embedded marketplace was dominated by people who were using eight-bit micro-controllers with relatively small amounts of assembly code. So going and telling them that they needed a 32-bit processor, or even a 16-bit one, with a high level programming language that they'd never seen before, was a significant barrier to entry! And certainly, on that front, I would do things differently—because we tried to tell people that they shouldn't worry about the instruction set, because they could do everything from a high-level language. Well, even still, there's a culture in many parts of the embedded system area, particularly the low end, where people still expect to program the thing—and certainly to understand the thing—at the level of the basic machine instruction set architecture. So for them, this is not the right approach to the market, and I think, generally, we probably weren't careful enough about the way that the product came over to the markets we were trying to address it to. So it came over to the propeller-heads very easily, they loved it, and the research community, and the people who were prepared to push the boat out and try something new, thought the was wonderful.

Krewell: Did you try approaching the military market?

May: Oh yeah, we ended up—well the military was very interested in it. The only problem is that that involved, would have involved us getting military spec processes and things run up. So I'm not sure if we sold to—we certainly sold to military research. I don't know that anything much went into military deployment. Not sure.

Krewell: And, in fact, that market has changed, too, because a lot more, what they call, COTS [Commercial Off-The-Shelf], off the shelf, they use today. But back in those days, they were very strict about standards and manufacturing process.

May: Yes, I certainly remember going into one or two big US companies with my guys to talk to people there.

Krewell: And then what—when did you decide to leave the INMOS?

May: Oh, well INMOS effectively became part of Thorn, and then it became part of ST, and we'd done the transfer, we'd done the T-4, the T-2, the T-800 was an interesting design, then they tried to do this T-9000 design, which was very interesting, and became far too elaborate, and—

Krewell: Was the T-9000, what was the target for that, was it—

May: Well, that's probably a good question. <*laughter*> So essentially, it should have been a follow on from the T-800, and it got into the hands of a rather over-enthusiastic development team, in my view, which actually tried to speed it up in ways that took them into a level of complexity and design that meant that the cost and time scale to design, it was just too long. And also the question of, what exactly was the market. Having got into this relatively high-end, embedded/supercomputing market, that's probably the place where the T-9000 could have made sense. It had virtualized all of the communications, so again, these days, on chip networks essentially automatically route packets from anywhere to anywhere. That was all part of the T-9000 design and, in fact, the piece that I probably had most involvement with was the communications part of the T-9000, because we built this thing called the C-104, which was a packet switch, which was actually finished in 1991, or 2 or something, so it was very early. It did dynamic packet routing, looking at [message] headers and forwarding packets with worm-hole routing on 32 100 megabit channels, so it was fairly state-of-the-art at the time it was done. And that was meant to be a switch that would go along with the T-9000, that would support directly in hardware, and microcode and stuff, all these virtual channel connections. So it would have been a great component, I think, for that kind of market, a much more—sort of—general purpose parallel computing component, from which you could just build up large arrays of them.

Krewell: You could easily see it today as a network processor.

May: Yeah, well you see, you could probably do some pretty good network processing with it, yeah. I, at that time, decided that there was going to be a pressing need for ST to have a system-on-chip technology, because what I could see was, there was a company I found myself in, with huge amounts of intellectual property. They had components and stuff that could do almost anything, from automotive to communications to whatever—and they were increasingly trying to put these things together, but you kept finding that, whenever you tried to put things together, it was easier to redesign them than to put them together, because they hadn't been designed to be put together, because they were all coming from different teams. And so I put together this proposal to build a sort of system-on-chip architecture, with a standard interconnection technology, that would make it possible to assemble systems quickly, and that's what we developed. That was called the Chameleon program, and that progressed quite well, but I had been in the company for 15 or 16 years, by this time, and I was beginning to decide that five or six architectures was enough for one lifetime! I got approached by Bristol University, asking if I wanted to become a professor again, and—I didn't really consider it for a while, and I thought, actually, this might be quite interesting. And so that's what happened. So in 1995, I moved back into the academic world.

Krewell: What was your title at INMOS, at the time?

May: Oh, I think when I left ST, I was called, Head of Advanced Microprocessor Development or something, but I was never really one for titles. It was sort of—I've tended to go through most of my life, luckily surrounded by very clever people. And you know—

Krewell: Well you're a rather clever person yourself. <*laughter*> And eventually, you—and I guess after a bit of academic, you wind up starting XMOS.

May: Well what happened, when I went into the academic world, I looked around, and I thought, actually, I probably ought to be the Head of this department because otherwise I'm just going to end up having arguments with the Head, who didn't really want to do the job anyway, so that's what I did, and I enjoyed this immensely, because I changed, basically redesigned the curriculum and, you know, reoriented quite a lot of the research activity in Bristol, and we grew quite a lot. I mean, it was too small, really, at the time I took it on, which was an attraction to me, actually—I quite like building things, whether they're computer science departments or chips, so—and one of the things that I'd done in the—along with that process, was to look at the whole nature of the way the project work was done, with the students, and, at the time, a friend of mine dropped in, and said, "Why is it that all of the American students, especially MIT and so on, have business plans in their pockets when they leave, and yours don't?" And I said, "Well, that's an interesting question." So I thought, that is an interesting question, so I actually got put into our degree program, a part of it which was that—alongside their final year student projects, they would see this thing as building a prototype and a plan—and I got somebody to come along and tell them how to write the business plans, and that's what we started doing, and they still do the same thing now. Along the way, one of my students who had attended my advanced computer architecture class, came along and said, "I'm looking for a project, and I want to do something in architecture." And so this was in 2004, I think, and I'd actually toyed with this idea that became XMOS, which—the notion is quite similar [to the]. You know—a potential fabric for building chips is to use an array of processors, and you can program them to do particular parts of applications and have a fabric that will allow you to use a single design—hardware design—to roll out lots of different product designs. And so I drafted this, and forgot all about it in 2001, when dot-com wiped away all the investment and so I thought: I'll leave this for another lifetime! Anyway, Ali turned up looking for a project, and so I said, "Well, here, let's try this one."

Krewell: Pulled it out of the stack.

May: More or less. <*laughs*> So he did a great job—he built an emulation of this thing on an FPGA, and he very cleverly, as he was leaving my office, he said, "What am I going to do about software for this thing?" And I said, "Oh, that's all right, I'll knock you up a compiler." I've never been so skillfully managed by a student in all my life, so every so often, I get a little nudge saying, "How are you getting on with the compiler? Oh, by the way, found a bug in it, can you fix this?" But we did—we worked very well together, and he wrote a good plan, and so we raised a little bit of money and kept him going, and then put the business plan for XMOS together, and set XMOS up, so that was what happened—but I'd always had it in mind that—you know, it would be good to be in the university again. I like working with young people and stuff, and students and others, and I thought maybe I'd start some new stuff up, and that's the one that happened.

Krewell: So how is the balancing between the startup and the university?

May: Interesting, but I mean, actually, it's—I stopped being Head of department, which was a great move, because it's normal to only do the job for five years or so, anyway, at Bristol, and I'd done it for ten by the time I stopped, so this was OK. So I tend, these days, to spend—most of what I do in the university is much more strategic stuff, and I still teach final year students about computer architecture, and I teach first year students about the history of computing, which I love, because actually, I remember quite a lot of it, and they're just great fun to work with, because you have these strange discussions. You say things like, you know, how did people work before the mobile phone? And they look blankly at you, and say, well, they did have telephones, didn't they? And—

Krewell: Try to explain to them, paper tape.

May: Yeah, I've done that, I've done that. I've done that, I've explained to them paper tape, I've explained to them relays, and all kinds of things that have gone out of the vocabulary completely. And so that's what I do mainly on the university side. And then on the XMOS side, of course, I still do a lot of this kind of technology stuff. I mean, I still program, for one thing, I wrote the original compiler for the XMOS thing. Not the production version, you don't want me for that. But—

Krewell: So, and what do you recommend to the students now? I mean, what is their big challenge in starting their own business, and do you think this entrepreneur spirit needs to be passed along to the other generation to keep going?

May: Yeah, I mean, I— the interesting thing is that when you—there are various things I do with the students, which are like that. With the first year students, I get them to write me very short summaries of things that interest them about historical topics that we've been talking about, and I make sure that they write short, carefully written pieces, just a page, actually, in most cases, and I make sure that they research them properly. I say I don't want a Wiki page, you know, I want you to go right back to the source material and check it's all true before you write it down. And I say, it's all online, you can access the journals and everything, and just go and see, get the words from the horse's mouth, make sure. Otherwise you'll find yourself endlessly replicating the same mistakes that are all over the web. So, and of course, what happens is, and writing business plans and thinking about things that way is another one of these sort of endlessly useful skills, because no matter what you're doing, and whether you're going to become an academic and try to pitch a research project, or pitch a thing to the investors, or pitch a thing inside your company, it's all the same thing. So I think students should be able to do that, and the answer to my friend's question was, the only reason that UK students don't have business plans is that we haven't asked them to— you know—and told them how to do it. There's no difference, apart from that.

Krewell: Well how is the funding, in the UK?

May: What, investment funding?

Krewell: Yeah.

May: It's not good. It's—but it's pretty messy everywhere for chips at the moment. They've all got lost in this bizarre, you know, web stuff. I mean, some of those valuations, I—

Krewell: Did you see a lot of this web stuff as a distraction for building real things, or is it, do you also see that as a lot of value to these web businesses that are showing up?

May: Well the question of what is value is interesting, isn't it? What is of value is what people are prepared to pay. But I mean, there is no inherent value in most of those businesses. I mean, in my view, the current state is roughly where we were in 1999-2000. It has to be a bubble. There is no way you can associate multi-billion dollar valuations with web things that have never turned a profit, and may well never turn a profit either. <*laughs*>

Krewell: Well is this a problem for the industry? Because I mean, it's difficult to get funding.

May: I think it's a big problem. I think, at the moment, and it scares me a bit, and it goes back to something I said earlier, that to do something real, that is world changing, is at least a ten year project, not a two or three year project. The only thing you can do in two or three years is to create bubbles, and we seem to have got an investment community that is trying to create bubbles. The question is, how quickly can we make this thing appear to have much more value than it has, and I choose the word, appear, carefully, as opposed to having much more value, and so you know, at some point, people are going to have to get real, I think. I just think we have to find a rather different climate for doing the investments, and just take a longer view. If that means that governments will have to do it, then governments will have to do it, although governments seem to take a fairly short term view of most things.

Krewell: Well, but what happened, in the United States, we've had the DARPA program, which is—

May: You're very lucky for having the DARPA program. We don't have that luxury. I'm often telling my friends that, that DARPA is—for the US—has been a great way of taking stuff from, sort of out of the lab to a point where it's a platform of some kind, whether it's going to be a platform for more research, or for a startup or something. And we would have to find other ways of funding that, and that makes it very difficult, because it is either very early stage seed funding, or an angel or whatever.

Krewell: Where do you see the opportunities still lie in semis, or in terms of where we still need more breakthroughs, where we need more investment?

May: Well, we need some breakthroughs in energy efficiency, and efficiency in general. We've had a rather strange ride over the last 20 years or so, and I think the—a lot of what's happened has been, well, it's been driven by what you can put inside a PC, and sell. And as long as it still worked from the power outlet, it was good enough. And what it's generated is a large amount of technology which is pretty much irrelevant for building, or is the wrong solution, to building a lot of the really mobile stuff that we've got. And it's the wrong solution to building the cloud or the supercomputers, too, where, same challenge, you're driven now, much more, by the energy that you have available, than by the silicon area or something. So the world seems to me to be changing quite rapidly, actually. I've been saying to people recently, I don't care what the area is, that doesn't cost anything. What costs is the energy in the system, and that's a very different kind of tradeoff. So that seems to be an immediate challenge, which is going to keep us occupied for quite a while. It certainly has caused me, of late, to rethink yet again, quite a lot of the ideas that I have in computer architecture. And—but it goes all the way up and down. I mean— what do you do, what are the process technologies, and does it make any sense to go and push to finer and finer geometries, or not? It's not clear. Or are we looking for a major shift into something quite different? But then all the way up, the same question arises. What about software? What about all that badly written software that's burning MIPs, but nobody ever bothers to rewrite—and they don't think they need to rewrite it, because historically, you just wait for Moore's Law to catch up and put more memory on, and it's OK. Well, it isn't.

Krewell: We'll start to wrap things up a little bit. What do you think has been the impact of the transputer program and INMOS, on the industry, and maybe even, specifically, in the Bristol area?

May: The impact in the Bristol area is pretty dramatic. I mean, essentially, there was INMOS, then acquired by Thorn, and then ST, so ST is still there with its design center. That was how things were until about '94-5, when Infineon decided to put a design center in Bristol. And then, one or two others joined the club. We had Panasonic there for a while, I think—they've probably pulled out now, but anyway, they were in there. Then we started to get start-ups. Element 14 was one that actually came out of a design team that was part of my original Chameleon project in the early '90s, and they joined forces with the residue of Acorn, which was the thing that spun out ARM, originally, and [they] created Element 14, which was sold to Broadcom. So we have Broadcom in the Bristol region now. Then the same guys went on and founded Icera, which has just been acquired by Nvidia, so Nvidia will now be in the Bristol area. And now I think there's a lot more. There's Picochip there still as an independent, doing arrays of DSP processors for base stations and things. So what's happened is that the place has become something of a magnet for design groups of either big companies or new starts. Obviously the people—almost for sure—some of the people from the more recently acquired companies will spin out and start new stuff up. I mean, I know of several chip startups in our university incubator at the moment, so it's going to keep on happening. So it's created, probably the—it's a very sophisticated group, because they're all brought up on processors, and that's the INMOS legacy. So what these people do, they all tend to be doing clever embedded processor designs, often in conjunction with communications, because there's a fairly strong communications expertise in the Bristol region as well. So that legacy has been very interesting to be part of, actually, and it just seems to keep growing. The impact on the industry more generally, is quite difficult to assess. We educated a lot of people, for starters, and I don't mean just in the sense of the people who came through the design teams, but also in the sense of where our products were used in the university groups and the research groups. All the time, I keep meeting people these days who say, I used your stuff in 1980-something or other. You know, they were doing computer vision or graphics or, or, or. Or simply because somebody has decided to run a class teaching concurrency, using the INMOS stuff. And they're far and wide. I think, at the height of the, sort of, transputer community, we had seven or eight thousand members, in about 35 or 40 countries, so you know, it reaches everywhere, pretty much, and so there are a lot of people out there who have actually got a level of understanding of the kind of concurrency and stuff that we did with the transputer. And actually, there were a large number of them who would like to see it back again. They basically say, can we have it back?

Krewell: Well is there any thought to resurrecting the transputer in a—putting lots of s onto one die or—

May: Yeah, I think it would be a great thing to do. I mean, actually, number one, the XMOS thing is actually quite close [to the transputer], but number two, and this will be the really interesting one to do, and I keep having plans to do this, and I haven't quite got there yet, would be to just create a completely open source transputer thing, because all the patents are expired, so nobody's got any particular hold over this thing. It's not difficult to implement—I mean, I had another final year student who wanted an interesting architecture project, and he took the transputer design. I conveniently still have copies of all the microcode for the , even the heavily thumbed working document that Roger Shepherd and I put together, which actually means that you don't have to reinvent that tricky piece of scheduling microcode. And he put this—he put a transputer together on an FPGA in a final year student project, and got it running. So you know, it's very simple. So you could imagine a website which has got it all downloadable, hardware designs, everything, and so—particularly now there's this upswell of interest in open source hardware as well as software—this would be a really neat way of doing it. The other hunch I'm beginning to have about the transputer is that actually, in terms of delivering an energy efficient technology, it's probably quite good. I mean, having recently measured where we're spending power in the XMOS designs, an awful lot of the power associated with every instruction execution is going into the register file access, not the arithmetic unit and things. So cutting down the number of registers to the minimum number that will do the job might make a lot of sense again—so these are some of the things I'm sort of interested in investigating. So I think the transputer will be a great candidate for doing an open source design and getting some interest around it. It's also—the other interesting thing about these very simple designs is that you could imagine doing a very optimized design. We do so much stuff by synthesis these days, and we lose a lot, because a processor is something where we know a lot about how to lay it out in a very structured way, so everything is kept very close together. And a lot of the power, of course, is lost in interconnect. So if you can actually do something simply enough that you can— more or less—hand-craft the design, you're likely to win. So I'd like to explore that again.

Krewell: Well it's a good—a good idea never goes away, it's always a good idea.

May: Yeah, well because I mean, when my students ask me about textbooks and things. I say, well, number one, don't trust what's in them. Number two—number two, one of the best—

Krewell: You haven't written many textbooks, then.

May: No, I haven't written textbooks. I don't use them very much, either. I mean—

Krewell: Really, not even the Hennessy and Patterson?

May: I don't use the Hennessy and Patterson book much at all. Actually, there are parts of the Hennessy and Patterson book I do refer them to. The problem I have with the Hennessy and Patterson book is that it essentially it starts off as if the whole world is RISC, and this is the only way to do things. And I don't like that, because it essentially—my universe, in terms of computer architecture, covers all kinds of different stuff—and there is no right answer. There are right answers for different applications, different process technologies and whatever, and the book I like best of all, actually, is—for getting a great overview of computer architecture—is Bell and Newell, the Gordon Bell and Newell book, "Computer Structures, Readings and Examples." But the only problem is that they haven't updated it since 1971.

Krewell: It's a little out of print, that thing. <*laughter*>

May: So we really need somebody to create the same thing from 1970 to now. Because essentially, there has been some new stuff and new ideas, but what you get, up to '71, is a wonderful summary of techniques. Because, remember, within that [book], are things like the [IBM]360-91, which is a super scaler, out of order processor, so all that stuff which is modern RISC architecture, is in there. There's CDC, there's vector processors, there's the PDP-8, the minicomputer stuff, and the stack machines, the B5000 and so on. So it gives you a great overview of different architectural techniques.

Krewell: I don't think it had VLIW [Very Long Instruction Word], though, did

May: I don't think there'll be VLIW—so there are—that's why I'd like the one that goes from there to now, because that's really what you want as a computer architect. I discuss [with the students] sometimes what the students think the subject is about—because often people think there's something deductive about it, but half of it is not deductive at all. I mean, you know, you have to throw ideas around—it's a very creative process—and you try ideas out, and then you start doing the work, and figuring out what's going to work.

Krewell: From your years at INMOS, and working on the transputer, do you have any other—another good story that always comes to mind?

May: How many stories do you want? I've probably got plenty, if you—

Krewell: Well get more of the stories when we have a team of you guys together, but the Mandelbrot one was a good one. Is there anything—

May: Well, I don't know, there was a—let me think. There's another cultural one which is quite funny. So when they—when I told you about this design system, and inventing this thing—so they went through the process, they built the sticks thing, and then they found they couldn't control this thing properly. The sticks compiler was always an issue, because you press the button and it tries to compress the design, and it gets the wrong answer. So then they started trying to make something that was more guided, where you could interact with it more, and control what it was doing, and they invented the friendly sticks editor, and they called it, Freddy. And then the next thing that happened was that they came up with a brilliant idea, OK, which was that they replaced these stick [thin] sticks, with fat sticks so, because with fat sticks that showed you what the actual geometry of the layout was, then you could specify and control, then the synthesizer could do a proper job. So they ended up with a fat sticks editor, and they called it, Fat Freddy.

Krewell: Of course. <*laughter*>

May: Of course. So you know, that team had quite a lot of fun. I mean there was another—the other classic, having gone through, done all this clever stuff, with building design rule checking into the CAD system, one of the first groups to do it, they got to a final stage, working quite late, just before the tape-out, as ever, and there they were with this stuff. And then somebody decided to put his initials on the corner of this thing. They would put the design team's initials in. But of course, as time went on, more and more initials kept appearing, so more people, they realized, were involved in it. And eventually, the guy who was running it, just put in the end, put, UTC&A, Uncle Tom Cobley and All. And of course then they pressed the button, and everything went sailing through, and then the whole lot failed design rule check, because they'd forgotten that all these structures they were putting in there with their initials on—

Krewell: Which had no connections.

May: No connections. <*laughter*>

Krewell: Yeah, this stuff doesn't fit anything, it's not doing anything, we should eliminate that.

May: Oh dear.

Krewell: So what about one other thing. What was your biggest ah-ha moment, when something just sort of clicked.

May: And I saw that amongst your questions, you know, I've been reflecting on that, and I don't know. I have—

Krewell: There was never one of those things where you just woke up and—

May: Well the thing is, that this is the way my head works, I sort of—I'm very conscious of it these days, and I suppose I've learned this about myself. I can easily get very absorbed in a difficult problem. I've got a pretty good sense of judgment as to what problems are solvable, just, and what problems are not solvable. And this is the other thing I try to get my students to think hard about. Picking off easy problems is pointless, because you're never going to change the world picking up easy problems. That's just boring stuff. So picking up impossible problems is an equally bad idea, because you know, because you’re never going to deliver anything. So the trick is to pick the things that are at the edge of the possible, OK, and so this is what I tend to do, and I've done it subconsciously all my life. I can sort of see what would be—and of course, also think about whether it's an important problem to solve, you know, so what happens then is that I get very absorbed in one particular problem, and I can have it in the front or back of my head for days, and then all of a sudden, there will be an a-ha moment—with a bit of luck. Sometimes it can take ten years, if it gets parked right at the back of my head and has to be re-awoken— and that does happen, too. It just happens with these things you've got stuck with, you know, you put them aside for a long time, then something happens, and you suddenly see that by putting two ideas together you can solve a problem. But quite often, what will happen is that, after a day or two, or a few days of thinking about a problem I've taken on board, the answer—the solution just pops into my head, at random, in the garden or in the shower or something, and I'll go into the office and say, I think this is how we should do it. And they're very odd, because some of them—I've said—people talk about inventing things—but I think—no, I think I discovered it. You know, it feels more like a discovery than an invention, because you look back on it, you think, well it's obvious that you should do it that way, but then, of course it wasn't. But there's a stream of these. Like I mentioned, this prefixing scheme for the instruction set. That was a thing I'd banged my head against—how to solve this problem. I knew it was—this whole business of making life easy for the compiler and efficient in the machine is an instruction encoding problem, and a lot of them are quite difficult to solve. And that one came out of essentially the same process. I had struggled with—I used the idea in a different context, I started struggling again with the same problems I'd put aside [previously], and all of a sudden, there was the answer, the fusion or these two things. There's been numerous similar ones—if I counted them, I must there must be dozens of those as we went through the transputer design, and even in the—and then the most recent thing, the XMOS thing, I can count half a dozen of those as well. I mean, if you look through my patents, they tend to correspond roughly to— <*laughs*>

Krewell: How many patents do you have?

May: Well it think there were about 35 that were associated with things in and around the various ST and INMOS projects. There's about another 15, most of which are sort of halfway granted in the XMOS thing, so there's probably round about 50, there will be soon. So—sort of ticks along—

Krewell: So how would you summarize your career?

May: Fun. <*laughter*> Well I've been very—I regard myself as having been pretty lucky, actually. I've been able to do things both in the academic context and in the industrial side. I've worked with some great people, I mean, really great people—in the people that I've been able to work with both on the hardware side, on the tools, languages and everything. And both young and old, and that's always very—it keeps you alive—and I think we've done some pretty interesting stuff.

Krewell: And then one last question, and that is, you're teaching computer history, so obviously you are a strong believer in the importance of computer history. Is there anything you want to say about that?

May: Well, I started doing it because I'm quite interested in history. I inherited an interest in book collecting many years ago from my father, and I have quite a lot of scientific books from, oh, the 17th Century onwards, more or less. And then I sort of got interested in the history of computing, and I have quite a lot of computer books, actually, also from the early days of computing. In fact, I have Allen Newell's copy of the Bell and Newell book, as it happened, which turned up out of the blue one day. So—and then I realized that there were a lot of things coming up—that the students just didn't really have much awareness of. So they'd ask questions about why things were like they were, and they'd assume that there was some deductive reason for this. And I think the first time this happened to me was actually in the company, where somebody said, why are strings in C terminated with zeros? And I said, "Oh, well that's just because of the architecture of the PDP-11," and they looked at me and said, "What's a PDP-11?" So I thought, a-ha, I've shown my age. <*laughs*> Maybe you should know. And of course you reflect on this, and you think, actually, there's no particularly good reason why strings in a high level language are terminated with zeros, and in fact, it's not a terribly clever thing to have done, from the point of view of the language designer, because when you try and actually detect that zero condition on a modern processor, the thing wants to scan the string word by word, so it's going to find the single byte zero somewhere in the middle of the final word, and know that's when to stop, so it's not an efficient thing to do on a modern processor at all, really. So, and there's of course, a whole lot of these. Why is this like it is? Well the answer is a historic answer, not because it's the way to do it. And so that was the thing that first set me going, thinking about that, and the more I've got engaged with it, the more interesting it gets, because we spend—I leave a reasonable time to actually discuss some of these topics, so along with things like, how did people work before mobile phones, you know, then you move on to, how did people work before computer networks. Well, they must have written letters to each other, mustn't they? Well, yes, but they didn't write a letter every three minutes! So some things are different. And then there are some other things, which actually go actually into another area that I've got very interested in recently—I ask them a question which goes like, how do you feel about depending on things that you don't understand? And actually, a modern 18 year old will answer that question and say, well you don't understand any of the things you depend on. You don't understand your motorcar. And I say, yes, but when I was your age, we did. So what happens when it all goes wrong, if it all goes wrong? What are you going to do, how are you going to live? You know, so you know nothing about the technology inside your mobile phone, but you just told me you're more or less dependent on it. So we have this kind of discussion as well—it's quite an interesting kind of thing to do. They're quite—they get quite interesting to talk to, but they haven't answered that question for me yet. And I am intrigued by it. There's a very nice, almost become a cult Sci-Fi story that you can find on the web, "The Machine Stops." E.M. Forster. So E.M Forster was, of course, the same author as, "A Passage to India," and also other things. He was actually a fellow of my college in Cambridge when I was there, King's College, which was where I was in Cambridge originally, which, of course, has been the home to quite a lot of computer scientists, including Turing, and actually, I think the oldest one is Oughtred, who I think is known for the slide rule—so we've made a disproportionate contribution to this [Computing]. Anyway, the gist of, "The Machine Stops," which is incredible to me, because it was written in 1909, is that the world is living in, sort of somewhat underground, following disaster and stuff, in these sort of cocoons, which are entirely serviced by machine, and people are communicating via video conferencing and stuff, and never meeting each other. And basically the infrastructure starts to fall apart. See, and the question is, what happens next? And it's an interesting question, isn't it, what happens when your highly technological infrastructure starts to fall apart? Then, do we collapse back to nothing, or can we sort of pick the pieces up, but at what point do we pick the pieces up, because for sure, if something mysterious knocked out a large chunk of the UK's—of the world's fine geometry process technology, then there would be a big problem right now.

Krewell: Sure, back up, know the history and be able to recreate it.

May: So can you back up and by the way, why are we going down this path of becoming so dependent on technologies, which, in many cases are—you couldn't call them necessary, I mean, a lot of them are—most of the iPods and things are sold as fashion items, not because anybody needs them. I mean, half the kids don't know how to use them, or at least, they don't know how to use most of the features on them. So there's a sort of interesting thread there, which goes back to my ultimate desire to build simple stuff, rather than complicated stuff. My whole culture is looking for the simplest solution to all these problems, and I'd like also to do things which, to a large extent —demystify. I mean, I think people, you know, if they're going to become so dependent on these technologies, they should know about them, they should understand how they work, to some degree, even if they couldn't recreate them themselves.

Krewell: That's good. And then how would you describe the importance of something—an institution like this, the Computer History Museum is a—

May: Well I'm looking forward to finding out what it does.

Krewell: It's the practical implementation of your ideas.

May: Interestingly enough, I've just become part of—I've just become part of the board of a science exploratory that we have in Bristol, and one of the things I'm discussing with them is, to what extent it should involve itself in computing related things. And of course we have, as somebody said, an embryonic counterpart of what you're doing here, over in the UK, who contacted me very recently—Bletchley Park, because Bletchley Park was the home of the code breakers in World War II, and the house and some of the infrastructure is still there. And they're turning that into a UK based computer museum. And I would like these—I don't know to what extent you do it, I'll probably find out soon.

Krewell: Yes, you will find out momentarily.

May: I mean, I'm very interested in getting people involved in building stuff, and I actually think we're on the verge of a very interesting revolution, because I think the combination of the kind of electronics we can do now, certainly some of the stuff that I've been doing, which I think is easily usable, and Arduinos and various of these things—they put a lot of computational intelligence in the hands of people, almost on their kitchen tables, effectively, and then we're about to do rapid prototyping, of course, which, I mean, that's taking off apace, which means that people can essentially invent all kinds of stuff, and build prototypes, stuff, at a very low cost. So there are lots of interesting things you can do without having masses of investment cash and so on. So we're always—you can almost imagine a rerun of the kind of things that went on that created the PC in the 1970s, but they're now based around a much more general kind of artifact, because there's more [we're able] to do, leaning towards robotics, so products that you and I carry around with us. And I think that involving particularly the young generation in that stuff is exactly what we should be doing, because they're the ones that will pick up on what this technology can do, and find the way of really getting value out of it.

Krewell: Yeah, there's actually a movement in the United States called the Maker movement.

May: Absolutely.

Krewell: Maker magazine, you know, sponsored by O'Reilly Associates and they have some really amazing shows, conferences, or fairs, where they bring people together who are building stuff. They have—they show kids how to solder, and they have a whole set—exactly the kind of philosophy. And then on Google, they have these kits where you can buy all the little pieces together and build stuff with it. There's a lot of that going on.

May: Yes, it's fantastic, isn't it? We're finally getting there, because as I say, I was lucky, you know, I had all this stuff being—basically chucked out from Second World War storage, and this was incredible machinery in some cases, which was being sold for nothing, more or less, and that's what I learned on, and I'm fairly sure that a significant chunk of my head was influenced by that, because you learn from these things. You look at these things, you pull them apart and you ask yourself questions, why was this designed the way it was? You know, and then you find the answer, and so this way, you know—I said I found my way through the educational system on the back of maths, but I was never going to be a world class mathematician, but it was jolly useful for getting through the educational system, but actually, you know, what I'm really good at, a lot of it is this sort of spatial stuff, and I must have learned a lot of that. Obviously I must have been innately quite good at it, but I must have really honed a lot of those skills on playing around with bits of equipment. Yeah, one of the fantastic items, I still have one of these in my basement, there's a computer that was developed for installation in aircraft in the Second World War, for aligning—for bombing, because it—basically the bombers had to just trundle along steadily, and of course, you know, get themselves in a good steady trajectory, and then drop the bomb. But of course they were sitting ducks for anti-aircraft fire, so this thing was designed to actually deal with allowing the plane to be banking and so on, and still compute the correct release point. It's unbelievable, it's based on wheel and disk integrators and function generators that are grooves cut into cylinders and all this kind of stuff. I spent a long time trying to figure out how on earth this had come to be designed, because it was designed at the outbreak of the Second World War, and it was in volume production, thousands and thousands of them, by 1941, 42. And it turned out, it was designed by a physicist who had, who subsequently became president of the Royal Society of London, and he'd been an advisor to the—he'd been working in the Cavendish Lab in Cambridge, and therefore obviously knew about all this kind of integrator technology that was in the mathematical labs and stuff, and he was a military advisor at the time, and he said, "Oh, I think I can solve that problem for you." And he sketched the design of this thing, it's just extraordinary. So that was interesting, and he was also a Kinsgsman, actually, so you know—yet another of them from the same college.

Krewell: Well I've run out of questions for you. Have you any other—

May: Have we covered everything?

Krewell: I have done the entire list of questions.

May: You need to know the people.

Krewell: Yes, well actually, I wasn't sure if you wanted to do that on a separate thing, where we—

May: That's fine, no problem.

Krewell: You want to—because have you thought about all the people involved, that—

May: Well I mean, the only—the essential question is how broad you go. You know, because it could be, yeah, I've thought of quite a few people, I mean, I know the guy that—I know Gerry Talbot is the phase-locked loop guy. Jonathan Edwards was very much involved with that ROM—RAM design. Guy Harriman, and John Beecroft, were the top level microcode people. Miles Chesney, was—and Eric Barton, and Jim Cownie were the CAD people, so we've got plenty of names here. Iann Barron is probably still around. I haven't—

Krewell: And he's the business guy who put it together, but also provided some of the—

May: Oh yeah, he's technically strong as well, yes, yes. He had a fair amount of technical vision, to put it mildly.

Krewell: Well how many people are still in Bristol, do you think? Is Iann still around Bristol, do you think?

May: Iann is still around in Bristol. Roger Shepherd is the other one, of course, who did a lot of that microcoding work. He's still around in Bristol. He still works at ST, actually. Yeah.

Krewell: What we might try to do is set up a video conference where we can get you guys together in a room someplace in Bristol, and then have somebody here in the United States interview, maybe, or one of our archivists, or I can get Gordon Bell or somebody else like that to perform the interview. Actually, we maybe—if we can get the funds, maybe we could fly somebody over there and do it there.

May: All right, all right.

Krewell: But they'll like to get names of the team, and then we'll kind of whittle down and see how—

May: Yeah, get a sensible number, otherwise it could get out of control.

Krewell: It gets out of control. We, just historically, from what they've done here at the museum, about five people on a panel is about the right size. So in some cases sometimes we split it, like, there's a business panel, there's an engineering panel, and there's maybe a software panel, and sometimes we do it like that. But seeing how so much of this was, the software and the hardware were so tightly coupled, but then there's also the hardware and the process guys.

May: Yes, yes.

Krewell: So interesting meshing of all of these teams.

May: Yes, yes, it was an extraordinary project.

Krewell: Yeah.

END OF INTERVIEW