Left to right: Andy Hertzfeld, Chris Espinosa, Joanna Hoffman, George Crowe, Bill Atkinson, Jerry Manock.

An Interview: The Macintosh Design Team

The making of Macintosh

On October 14, 1983, the design team for Apple Computer Inc.'s new Macintosh computer met with BYTE Managing Editor Phil Lemmons at the company's Cupertino, California, headquarters. In the dialogue that followed, Bill Atkinson, Steve Jobs, Andy Hertzfeld, Larry Kenyon, Joanna Hoffman, Burrell Smith, Dave Egner, Chris Espinosa, Steve Capps, Jerry Manock, Bruce Horn, and George Crowe discussed the evolution of their brainchild.

BYTE: How did the Macintosh project begin?

Jobs: What turns on Andy and Burrell and Chris and Bill and Larry and everyone else here is building something really inexpensive so that everyone can afford it. It's not very many years ago that most of us in this room couldn't have afforded a $5000 computer. We realized that we could build a supercheap computer that would run Bill Atkinson's amazing Quickdraw and have a mouse on it—in essence, build a really cheap implementation of Lisa's technology that would use some of that software technology. That's when the Macintosh as we know it was started.

Hertzfeld: That was around January of 1981.

Smith: We fooled around with some other ideas for computer design, but we realized that the 68000 was a chip that had a future and had...

Jobs: Some decent software!

Smith: And had some horsepower and enough growth potential so we could build a machine that would live and that Apple could rally around for years to come. So we looked at what the Lisa group was doing and knew that the designers were onto some really hot ideas. They have a lot of very advanced things they want to do with Lisa. Mac basically does one thing at a time as opposed to doing several things simultaneously. The memory-management unit that's critical for a Lisa application, for example, becomes something we can do without very nicely. Our real goal was to design a great system with just a bit map and based on a 68000 but also a really cheap system. Could we write incredibly great software that wouldn't chew up megabytes of memory? To do what used to take megabytes in a very tiny machine?

Atkinson: It's not like we didn't want a memory manager in it or didn't want lots of memory or didn't want hard disks. What we wanted was for lots of people to be able to own these things. We saw something beautiful that we built and we said, "How can we get this out to a lot of people?"

Espinosa: It doesn't matter how great the computer is if nobody buys it. Xerox proved that. The key thing you've got to remember is that back then, if you told anybody you could build a computer using a 68000 with anything under a hundred integrated circuits, they would have said you were crazy.

Kenyon: Most people have twice as many chips just for central processing unit support on the 68000. So nobody had ever conceived that you could build a cheap system...an Apple II chip-count system with a 68000.

Atkinson: We want the most computer that you can get for the least dollars so that the most people can have it. . . and then you can concentrate on making the world's best software for it.

Bill Atkinson.

Espinosa: And you look at this board, and every chip on there is pretty expensive. There's not a lot of jellybean TTL [transistor-transistor logic] running around, not a lot of little off-the-shelf chips. Everything on there costs $4 to $9 apiece, and that's expensive for a chip. But we've got so few of them; instead of taking up board space with a lot of stuff that you just don't need and making it unreliable because you've got to have connectors and you've got all these problems with soldering. If you just carefully pick what chips you want to use and you've got somebody like Burrell who's genius enough to put the right ones together in the right way and make them do things they've never done before, then you can come out with something that's small and inexpensive and incredibly powerful.

Smith: What gets me is that a lot of programmers will have this long laundry list of things they must have before they will sit down and allow fingers to touch the keyboard. I was really lucky because these guys are the best programmers I've ever seen anywhere, not just with Apple. They walk around between the Apple divisions, contributing this amazing graphic stuff to Lisa—and then help out on the Mac, too. Everybody had this common goal of making the Mac

Andy Hertzfeld.

flexible and general-purpose because we didn't know what we might want to do five years from now. We knew the kind of direction we were going, so instead of building in a graphics controller that takes 25 chips and then trying to figure out some way to soup up the architecture so that it actually would work with it, we relied a lot on the processor assembly-language code in ROM [read-only memory]. And it turns out that we can make the whole system go faster by eliminating a lot of the bus traffic that normally slows the machine down.

Jobs: We learned a lot on Lisa.

Atkinson: We're still learning a lot!

Jobs: If you read the Apple's first brochure, the headline was "Simplicity is the Ultimate Sophistication." What we meant by that was that when you first attack a problem it seems really simple because you don't understand it. Then when you start to really understand it, you come up with these very complicated solutions because it's really hairy. Most people stop there. But a few people keep burning the midnight oil and finally understand the underlying principles of the problem and come up with an elegantly simple solution for it. But very few people go the distance to get there.

One of the things we really learned with Lisa and from looking at what Xerox had done at PARC [Palo Alto Research Center] was that we could construct elegant, simple systems based on just a bit map...no character generators...and save tons of chips if we had software fast enough to paint characters on the screen, given the processor. Apple was the first company to figure out how to do that with a microprocessor...and really still is the only company that's doing it with a microprocessor. That's what Bill figured out how to do with Quickdraw.

The real reason that we chose originally to use the 68000 was so we could pick up Quickdraw. Macintosh uses the exact same graphic structure and package, the exact same code, as Lisa does. So, by paying a little more for the microprocessor, not only were we able to give the customer an infinitely more powerful chip than, say, an 8-bit chip or one of Intel's baby micros, but we were able to pick up this amazing software, and that allowed us to throw tons of chips out of this thing. We didn't have to get special custom text or graphics chips. We just simplified the system down to where it's just a bit map on the screen, just Bill's amazing software and Burrell's amazing hardware, then in between that the other amazing software that we have. We tried to do that in every single way, with the disk and with the I/O...rather than slots.

When we first started off with Apple II, the variability—how you customize your machine—was with hardware; you plugged in a card. And because we didn't have any idea what these computers were going to be used for, that variability was very important. But now we have a much greater understanding of what people are using these products for. And the customization really is mostly software now. The way I customize my machine to do what I want is by sticking in a disk more than anything else.

Atkinson: We've already built in the hardware that most people want.

Jobs: Right. Most of the options on other computers are in Mac. So Andy and Burrell really came up with an I/O scheme that was serial. We don't have slots...slots cost a lot of money, they make the box much bigger, and you need a much bigger power supply because you never know who's going to plug in what. Do you realize that in an IBM PC the video board, just the black-and-white video plug-in card, has got way more chips than the entire Macintosh? Anyway, so the Mac's got most of the stuff built in. Rather than putting in serial ports that operate at 9600 or 92,000 bits per second, we paid more money and we put in this super chip. We used the Zilog SCC chip that Burrell picked out, and Larry Kenyon and Andy wrote the software to make this chip sing. And it goes up to, what, 230 kilobits per second?

Smith: It can go up to a megabit per second with external clock.

Jobs: And it does all the asynchronous and tons of synchronous protocols all inside the chip. So we've got superhigh-horsepowered serial ports.

Smith: The whole idea is that later on we'll be able to have logical slots instead of physical slots. We'll be able to have multiple devices per port; we'll use a port a lot like the way you have slots in Apple II. But one of the other advantages that Steve didn't

mention is that you don't have to change the memory map of the computer. Andy and Larry Kenyon worked on the system and the driver software and things like that. They said, well, gee whiz, on the Apple II you keep having the rug sort of changed on you; someone plugs in a new card or, worse yet, on other micros people are plugging in different software and different hardware, and it's hard to keep track.

Atkinson: You get into incompatible combinations; you just can't use this card with that card.

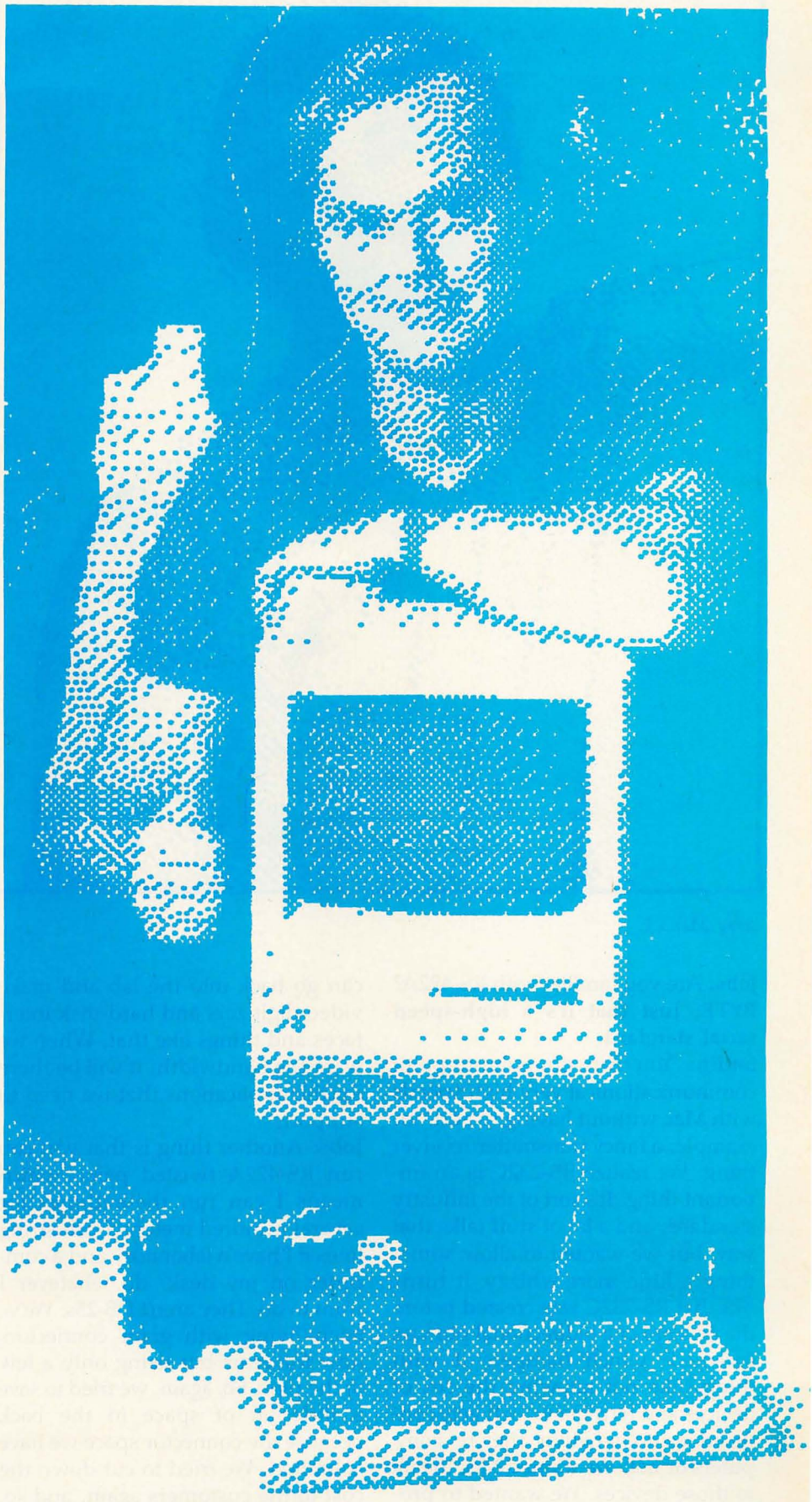
Jobs: The other thing about the hardware is that when Apple II was designed, a microcomputer system cost a lot of money to build. I mean, to get a microprocessor and RAM [random-access read/write memory] and ROM might have cost $50, $60. You obviously wanted to share that among the peripherals, which is what the Apple II did, what any slotted system generally does. Now you can buy a microprocessor and RAM and ROM in a single-chip micro for about $4. So giving each of the off-board peripherals its own little microprocessor system is adding $5 to the cost of the peripheral. And the cost for providing them with the bandwidth that's needed for most of the peripherals that are not on this board is very low. Add a $5 bill to the peripheral, put a single-chip micro in it, and then talk serially, rather than have every single user pay an extra few hundred dollars for the price of the slots that may never get used.

Atkinson: One way to look at the bandwidth thing is real simple: if there are 128K bytes, that's an eighth of a megabyte. There's 1 megabit in the machine, so the worst transfer you could think of, transferring the entire contents of the machine, takes one second. You transfer the entire contents of the Mac through that serial port in one second.

BYTE: What are the serial connectors?

Jobs: There are two connectors, DB9s...

Atkinson: They're tricky. They can run anything from 300 baud on up; you can use them as RS-232C or RS-422A.



Burrell Smith.

Jerry Manock.

Jobs: Are you familiar with RS-422A?

BYTE: Just that it's a high-speed serial standard.

Smith: You can do point-to-point communications at very high speeds with Mac without having to add, for example, a fancy transmitter/receiver thing. We realize RS-232C is an important thing. It's sort of the industry standard, and a lot of stuff talks that way, but we wanted to allow something a little more whizzy. It turns out that RS-232C was created before the concept of a bidirectional pin was invented, which hampers it with things like not knowing the sex of devices and terminals. . . . It gets confused as to whether they're computers or not. We wanted Mac to talk to those devices. We wanted to provide for the future so that, for example, if I ever get a spare moment, I can go back into the lab and make video digitizers and hard-disk interfaces and things like that. When we want the bandwidth, it will be there for the applications that we need to support.

Jobs: Another thing is that you can run RS-422A twisted pairs, which means I can run these things for several hundred meters. I can string lines if I have a laboratory and a computer on my desk, do whatever I want to do. They aren't DB-25s. We've been living with giant connectors now for years but using only a few of the pins. So, again, we tried to save a little bit of space in the back because the connector space we have is limited. We tried to cut down the cost to the customers again, and so, for connecting to devices like printers and modems, which we offer and which are the most prominent, we just supply the cables. We also will supply cables from one of these things to a variety of DB-25s. . . for the modem version, the printer version. . .

Atkinson: Lines 2 and 3 are switched on a modem versus a printer, so you just use a modem cable or a printer cable.

BYTE: From a very early time you knew that you wanted to take advantage of Lisa's software technology, and you also had the goal of making that possible at low cost. When did you have a consensus on exactly what this hardware would have to be to achieve that goal?

Smith: In 1981 we started looking at the Lisa. I came up with a proposal that said it ends up costing $14 more to use a 68000 with 64K bytes of memory than it does with 6809-based machines, if you count power supply. It turns out that it's actually easier to interface memory to a 68000 than to a 6809. So in January we started really looking at the 68000 and the work that Bill was doing.

In June of 1982 we finally decided on what we thought was enough video. It turns out that the original machine had 384 by 256 pixels. We chose that because we thought we had a shot at squeezing the machine down into 64K bytes, and we didn't want to throw away a quarter of the memory just for the screen.

Atkinson: The thing that drove us is the 80 columns. In a word processor, we really wanted the lines to break on the screen at the same place they break on the printer. There are two kinds of word processors. There are the ones where you just have a string of characters and you see them however they wrap on the screen. Screen wrap is a function of the screen, and how characters wrap on the printer is the printer's doing. Then there are word processors where what you see is what you get. You lay out a line and you know it's going to break at the same place on the printer as the screen, so you can do columns and tabs and a couple of columns of numbers. Then you have to have enough pixels to generate a full printer line across. We thought we could do it

The Wizards behind the Macintosh

Bill Atkinson *nearly had his Ph.D. in neurochemistry before he admitted to himself that his real love was computers. He "got a quick E.E." and started his own company. He was happily minding his own business when his friend Jeff Raskin asked him to come see what was happening at Apple, which was then six months old. Bill wasn't really interested, but airplane tickets showed up in the mail, so he took a look. What he saw was "several years reaching into the future" of anything he could do where he was. He stayed to write Apple's Pascal and later became Mr. User Interface for Lisa before he moved over to the Mac team.*

Andy Hertzfeld *says, "The Apple II changed my life." The computer people at Berkeley were a little narrow-minded about letting a grad student really get into the computer as Andy wanted to. So he spent nearly all the money he had in the world on an Apple II and had a computer he could control completely. He decided the Apple was more interesting than his classes and began writing programs for magazines. When Apple bought one of Andy's programs, Steve Jobs offered him a job, which he took when he finished school. He worked on silent-type printers and Apple III demos until a shake-up in his part of the company shook him loose. He looked around and decided to go with Mac.*

Larry Kenyon *arrived at Apple from Amdahl with a double degree in psychology and computer science. He was working on Apple II/Apple III products when the same shake-up that shook Andy loose freed him, too. Andy asked Larry to join the Mac crew because he was one of the few people who understood the arcane art of making the Apple II work with printer peripherals, and anybody who can do that has to be good. No one in the company really believed that Mac was a product when Larry joined the Mac team. It was just a research effort, and there was some risk involved—would you still have your job in a few months?*

Joanna Hoffman *is still on leave from her Ph.D. program in archaeology at the University of Chicago. She has a background in anthropology, physics, and linguistics. She came to Apple because of Mac. After using her computer skills in the field of archaeology for so long, she was tired of looking at the past and turned to the future. She was Mac's entire marketing department for more than a year. She wants to make Mac a tool that feels natural for international users by making it speak their languages.*

Burrell Carver Smith *encountered the Homebrew Computer Club in 1975, got hooked on microprocessors, and moved to the Bay Area. Just riding around in a borrowed truck one day, he saw Apple and decided to drop in. The only job Apple had available was in the service department, repairing Apple IIs. He took the job and fixed at least a thousand Apple II boards and got involved in other projects before Jeff Raskin and Bill Atkinson recruited him for Mac. He talked the Lisa engineers out of some chips and stuff and got a prototype running over Christmas 1979. He was the first full-time Mac person after Jeff Raskin.*

Chris Espinosa *says, "There was no life before Apple." At 13 years old he could be found cruising up and down the bus line in his home town, spending a few hours at each Byte Shop on the line until the owner threw him out. He discovered the way to keep from getting thrown out was to write demo programs for the machines, so he wrote for whatever was lying around—Altairs, IMSAIs, or this weird new machine called Apple I. His mom worried when he was offered a ride to the Homebrew Computer Club meeting with two scruffy characters named Jobs and Wozniak, but she gave in, and the rest is history. Chris spent a Christmas vacation debugging Apple's BASIC in exchange for a whole row of 4K-byte RAM chips, which he thought was a pretty good deal. He worked part-time during college writing BASIC programs and reference manuals and signed on full-time when he graduated. He likes being in on the design process—"If the machine is designed right in the first place, you don't have to write a lot about it."*

Jerrold C. Manock *was a freelance product-design consultant with a Stanford education who finally joined Apple when he saw that three-quarters of his billing was to Apple anyway. He worked on the Apple II, the Disk II, the III, and Lisa before designing Mac. In Macintosh, he says, "The outside matches the inside in elegant simplicity."*

Bruce Horn *grew up at Xerox PARC, much the same way Chris grew up at Apple, and later attended Stanford. Bruce started working at Xerox when he was 14 years old—he was one of the kids Xerox brought in to test Smalltalk. Turns out he was brighter than most and became a systems wizard who actually implemented Smalltalk on a variety of different processors. Bruce is all of 23 years old now, but he spent seven years at Xerox PARC and brought Apple that perspective.*

George Crowe *and* **David Egner** *designed the analog board in the Macintosh.*

Steve Capps *assisted Andy Hertzfeld with the systems software.*

with 384, and we tried it with real live documents—and we couldn't do it. You could do it with 512, but you couldn't do it with 384.

Smith: The diagonal lines look better, too; the jaggies are removed somewhat, and things like that. So, with that, we said, OK, what's that going to mean? And we ended up with 128K and...

Atkinson: 22K bytes on the screen, and in a 64K-byte machine you couldn't have afforded it. That drove us to 16 RAM chips instead of 8.

Hertzfeld: By then, we knew we were going with 128K bytes anyway, to run the applications.

Jobs: I just thought I'd show this to you. This is the IBM video board; it's only video, nothing else. It's 69 integrated circuits, more chips than an entire Macintosh, and it basically does nothing. And it doesn't even do that very well.

Espinosa: Forty percent more chips than the Mac.

Jobs: So that sort of gives you a feeling. And again, that just has the video on it. Macintosh, in addition to having video that's far higher in resolution and far faster, has a 32-bit microprocessor, 128K bytes of RAM, 64K bytes of ROM, two serial ports, the mouse, the serial, keyboard, and mouse interface, the incredible sound, the clock calendar, the disk controller. . .

Smith: We rolled the whole disk controller into one chip.

Hertzfeld: And it has Lisa's graphics and user-interface software built into every board.

Jobs: Andy was sort of the software technical leader behind the project, from its inception. As Andy puts it, software sometimes stands on its head to get rid of a chip in the hardware. And so, with a system as powerful as this, we wanted to take advantage of all the features, for instance, in the serial chip and the disk and stuff. We really wanted to be able to have the serial ports reading while the disk is spinning, while the mouse is moving, while it's making sound. You know, all with that single board.

BYTE: What were the roots of that operating system?

Kenyon: When we started, of course, we were looking at the work Lisa was doing, and the Lisa group was rolling its own operating system, and it just didn't seem appropriate. We took the graphics software, which was perfect for our machine.

Capps: The Lisa's operating system took a lot of the user interface. For the window manager, even the memory manager, we started with what Lisa had.

Hertzfeld: It turns out that Quickdraw is built on top of what Lisa would call the intrasegment memory manager. You relocate little objects. We took that because Quickdraw required that support, and we sort of turned it into our system-wide memory manager. Even the Lisa group uses it only for the intra-application memory manager. Someone mentioned a neat way to do a file system, and we thought about it and said, "Gee, that's a good way of doing it," and so we did. A lot of it was experience on the Apple II, knowing what was sort of bad there—what we wanted to do great here. That at least was the conception of the asynchronous I/O. I knew from the Apple II that when you make a disk request it waits there for a whole second, a million microseconds, just waiting for the disk to come up to speed. We should be able to do other useful work while that's happening. On the Apple II if you want to make a beep, the whole processor, the entirety of the machine, is devoted to making a beep. And when you've got all the horsepower of the 68000 there, you don't want to waste it all on making sounds.

Atkinson: We still make a beep with the processor.

Hertzfeld: But we time-slice the processor such that you can be doing other things. It happens on the interrupt level instead of being dedicated. Macintosh uses the processor for everything, just like the Apple II does. In terms of the disk, we have

the same disk-controller architecture as the Apple II, but we are just a little more sophisticated in how we use interrupts. We give the time back to the applications while the I/O is going on.

BYTE: Can you say more about the custom disk controller?

Smith: Sure. A long time ago we sort of figured that everybody who was doing designs at Apple with disks loved what Woz [Steve Wozniak] had done on the Apple II. I'll never forget, the first time I looked at the Woz controller I said, "OK. Well, this must be the interface disk controller. Where's the disk controller?" I never found the disk controller. And we've just been in love with the way that that's done. It's used to modify group code. One of the things we knew, though, was that disks would be going faster in the future. So we initially designed this chip so the whole company would be able to have an ultra-low-cost way of using Wozniak's disk technology for every product. But we knew that we weren't just going to be going at 4 microseconds per bit, that twice that would become an industry standard . . . at least an Apple internal standard. So we built in a mode, a high-speed mode, so that it can go twice as fast.

Atkinson: While you're getting input from the serial port at 19,200 bps, you can be writing to the disk and not missing a beat. It's not the buffer that's doing that. It's Larry Kenyon. Every 4 nibbles, you look to see if there's something on the port, because in one sector's time, 24 bytes go by.

Jobs: After we reexamined everything, including the disk format, we said, "Do we want to go to MFM [modified frequency modulation]?" And the more we reexamined it, what became clear was that the original idea that we had for a disk in 1978, which we are still using, is great.

Atkinson: We get 400K bytes on this thing, while most people get only 270.

Jobs: As an example, our scheme has twice the margin of MFM. In other words, when you're shipping a million or two million computers a year, which we intend to do, when people are buying media from 10 different sources and they expect to take disks out that were recorded in Alaska in really cold weather and stick them into machines in Florida in a heat wave and have them work, that margin is really important. If you want to equate that to reliability, we are significantly more reliable than any other disk system on the market, while having higher capacity. So that was the key decision, to stick with the same encoding format and the same scheme that we've used since 1978. So, while everyone else is running at roughly the same rates as Apple II, the IBM PC, and everything else, we doubled it on Macintosh. We set a new internal standard with the 3½-inch disk and this new single-chip controller. And every new 32-bit product at Apple will use that new standard. The media, the sector format on that media, the disk controller, and the routines and everything to drive them is a new Apple 32-bit standard that you'll see com-

ing out in every future product that we do in that family.

Smith: There were some voices within the company that said, "Oh, you guys ought to go with standard formats and things like that." We looked at doing that and it turns out that it takes more chips to interface to a standard floppy-disk controller, and we have...

Jobs: Well, I can go get the IBM floppy board. It looks to have about 45 to 50 chips on it...

Espinosa: I'll come and help you carry it.

Jobs: ...including an LSI [large-scale integration] disk controller—far less performance, far less capacity, far higher cost.

Atkinson: And less reliability.

Jobs: Oh, far less reliability. Larry's software senses the disk speed, and Burrell's hardware can adjust to one of four hundred speeds. So if it's written on something that's a little out of whack, we can just adjust right down to the necessary speed and read it. Everything on the Macintosh board—the serial timing, the disk timings, the microprocessor timings, the video timings, the sound timings—comes from one crystal oscillator and is synchronized from one source. And, again, it's better, of course, technically to do it that way. Everything works much better, but it also saves parts, and we can offer this thing cheaper to customers. And most of this stuff customers will never ever realize or care about anyway. I mean, who cares how many crystal oscillators you have? But you *do* care about how big your computer is. You do care about how much it costs, and you do care about how well it works.

Atkinson: If you ever drop your computer you find out quickly how many crystal oscillators you have.

BYTE: So with the variable speed in the disk drives, I guess there's no problem having two drives that are 3 percent different in speed.

Jobs: We read it and adjust it so that the speed is accurate relative to that crystal. That crystal on the board is superaccurate. We can adjust the disk drive relative to that superaccuracy.

Atkinson: You force all the disks to go at exactly the same speed by having the software constantly monitoring the speed and saying, "Ah, it's running a little slow; jack it up a little bit," so that each disk doesn't have to be adjusted at all. You switch disk drives, and the new one will run at exactly the same speed because you force them all to.

Smith: It turns out that the speed variations occur partly because you plug in a new cassette that loads the motor down in a different way and also because of temperature variations that cause very long-term drifts in the disk speed. Using a little bit of the processor to fix that doesn't cost us any performance at all on the system.

BYTE: What about the display electronics?

Atkinson: Where is the display controller?

Hertzfeld: It's hidden.

Jobs: If you bite into that IBM display board, it'll totally flicker if you do it at the wrong time. You've seen that, right? Woz just came up with this really brilliant way to do the Apple II. He realized that memory was about twice as fast as the microprocessor needed it and twice as fast as the video needed it. So he put the microprocessor over here and he put in essence the video over here, and he put some multiplexers in the middle. He shared the exact same memory between the two in a way such that this one thought it had all the memory all the time and this one thought it had all the memory all the time, yet they shared the same memory! All this thing had to do was write into certain memory locations and, magically, it would appear on the screen. The microprocessor never even had to think about the screen. All it did was look at memory locations.

Atkinson: And there was no way to glitch the video because accesses were mutually exclusive.

Jobs: Right. And so it turns out that, try as we might, we have never been able to find a better way to do it.

Atkinson: At the same time that the processors have gotten faster, memory's gotten faster; the memory is still twice as fast as the processor.

Jobs: And so, again, it gives you greater performance, because you don't have to write only at special times and slow yourself down. It cuts the chip count way down because you don't need two banks of RAMs, so the customer's not paying for these extra chips, and it just makes a more elegant product.

BYTE: How far does the similarity extend between the Apple II video and the Mac's video?

Smith: We have a three-part memory architecture on Mac. We have a DMA window for sound, video, and CPU...shared by three devices. Also, what we do that is a little more sophisticated than Apple II is return memory cycles to the processor during horizontal and vertical retrace. And with the analog design we're able to lengthen the horizontal retrace interval, which gives us more performance for graphics by making more time available to the processor from memory and giving the analog electronics more time to retrace the beam. On the Apple II, Woz sort of designed this logic board and the power supply was kind of added. On Mac, we really designed the entire system as a complete system from the ground up, so we used different constraints. I would say there's not much similarity. The great thing about Mac as a product is that it really wasn't designed as just this piece over there and this piece over there and this other piece...All of it was designed in parallel, everybody knowing what everyone else's job was.

BYTE: How did you decide on the appearance of the machine?

Manock: Our goal in the beginning was portability. We actually had this cardboard model that looked amazingly like the Osborne. And that was way before the Osborne came out. As I said, portability was primary here, and this version had an attached keyboard that had a sort of rubber boot around it that would fold up and give you protection over the screen. Steve really changed the emphasis of the product one day when he said that we didn't want portability to be the primary aspect of this, but we did want it to take minimal desk space. With that goal in mind,

we realized that the keyboard didn't have to be exactly the width of the computer.

Jobs: To use the earlier design you had to have some sort of arrangement to tilt it up. And what we noticed was, well, fine, what if you just lift the back up here like this? Then, because you have all this space underneath, you could put the floppy disk underneath. So you make a unit that's more vertical, has a smaller footprint.

Atkinson: It has to be up enough so your eyes can see it anyway; you need the height.

Manock: Steve thought, too, I think—in a gut reaction sort of way—that everybody was going low profile and wide, and we never have wanted to be a "me, too." I think our vertical format is correct when you think of human factors.

Hoffman: Jerry, you might want to turn the back around. We made it truly international. I think it's one of the few products aside from Lisa that is completely usable anywhere you care to take it.

Manock: Did you see the icons on the back?

Hoffman: We started out with the case and went from the outside in, trying to make it more and more international the more we thought about it. And Jerry was just great as soon as he realized that we really did want to bring it to the whole world. He had marvelous ideas on how to eliminate every word of text, take everything off the package so that we don't have to be an American product anywhere that we go.

Jobs: In Mac, there's no English on the outside of the case. Everything's iconic. And there is absolutely no English in the ROM. It is universal in nature. When the thing comes on it puts a few icons on the screen. If something goes wrong, it can't boot or something, it puts a frowning Mac on. If it's booting it puts a happy Mac on. It loads all the languages, all the country-specific stuff, off the disk. So, because the keyboard is detachable and mapped anyway, to localize Mac all you do is change the keyboard, manuals, and the disks. Nothing in the box has to change.

And another real breakthrough is this thing called Resources that Bruce Horn invented.

Hertzfeld: The data is factored out from the code. You know, most programs are a mixture of control logic and just raw code.

Atkinson: The virtual-memory architecture on the data parts of the program allows us to factor it out so that, without rewriting a program at all, without recompiling or relinking the program, I can take a copy of Mac Paint and in 15 minutes make a German version.

Hertzfeld: Because all the text is kept in a well-known, well-defined place.

Horn: Until December, people didn't really know what the resource manager was, because they really hadn't had any contact with it, besides me. I knew what I wanted from it because I had to do Finder and all that other stuff. Andy just looked at it over time and figured out what you could do with it. And I was trying to say, well, this can do this and this... It was really Andy having the biggest view of the system saying that this could really be a great thing for a lot of stuff.

Hertzfeld: Another thing to ask Bruce about is the Finder, which is our most important application, the first thing that comes up on the machine. That's the program with all the little icons, the desktop manager, I guess we're calling it. That's Bruce's conception and communication.

'In Mac, there's no English on the outside of the case. Everything's iconic.'

Hoffman: There are numerous subtleties with this. Picture a dialogue box, for example. A dialogue box, when you put English text in German, starts overflowing its limits and starts looking very different. You have a button that says, "Put this away." In German, that takes a paragraph and overflows the box.... But Resources lets us change not only the text but also the physical look of those dialogue boxes, or anything, through *something* called Resource Editors.

Jobs: Otherwise, you'd have to get into the source listing. You'd have to change not only the languages, as Joanna said, but also the geometries of the dialogue boxes and make them bigger. It would take you awhile; it's not something that's impossible, but it's something that never gets done. And it's certainly something that you have to be the originator of the program to do. What we've done by pulling all the language-specific stuff out, through this beautiful mechanism called Resources, is write these other programs called Resource Editors. By running a Resource Editor, you could, if you knew German, simply run a program on the program, get in there—literally on the screen—and just stretch the boxes bigger. You could select a text and retype it in German and move things around if you wanted. You can examine every icon, every dialogue box, every alert box, every pull-down menu, everything, without being a programmer,

without getting the source code, and very quickly, too, using the user interface of the Macintosh.

Atkinson: Anything that XYZ software company put together, even though the company didn't think about Taiwan, will run in Taiwan.

Jobs: But do we want it to run in Taiwan?

BYTE: Are you going to market it aggressively in Japan?

Jobs: Yes.

Hertzfeld: My favorite thing about Resources, being selfish, is that the same facilities that allow us to translate English into 7, 10, 20, a million different languages are the same facilities we use to translate technish to English in the first place.

Hoffman: The other component of this is that it allows us to not just introduce products that feel to the native user like a native machine, natural to them, but also that we can start coming very close to making simultaneous product introductions. The software that is developed in the U.S. can fly over there for them, for the fragmented markets in Europe, for example. Europe does not allow for the same kind of development of software houses as the U.S. because the markets are all so fragmented you can't amortize development of the software over as large a user base. But given that the Europeans now have the capability of using a localized, *globalized* software, if you will, their market grows because each individual software developer in France now can view the whole world as a market. We feel that it will give an impetus to the development of software developers, third parties, in Europe, and in more fragmented markets as well.

Smith: An international power supply, too, so the exact same unit basically can be used anywhere in the world.

Egner: It doesn't care whether it's 50-Hz input.

Manock: Just one additional thing on these: the icons on the back are from the International Electrotechnical Commission (IEC). We didn't invent all these ourselves. . .wherever possible we used symbols that already existed—for example, AC line power—that are world standards. Where we didn't have symbols that existed, we used the IEC's closest symbol as best we could and then added what we thought made sense. For example, we needed a symbol for a modem, so we started with IEC's telephone symbol. We tested them to make sure there was good recognition. We'll submit these new icons to the IEC to have it suggest that they be the standards added to its encyclopedia of symbols.

BYTE: What is this machine going to make possible that other comparably priced machines have not made possible? How will it change the personal computing scene?

Jobs: Right now, as you know, when you use a word processor, it will do two or three things. The first thing Macintosh will do is make the existing types of applications an order of magnitude easier and more approachable for people. Therefore the available market for this machine is going to be giant compared to the available market for the people who are willing to invest 40 to 100 hours learning to use their computers. That's the first thing.

The second thing is that there are going to be new types of applications available that could not be available on the current generation of personal computers—it is technically impossible to do. The perfect example is Paint. Paint is impossible to do on an Apple II or an IBM PC or any of the other first-generation products. You can do a mockery of it, but you can't really do it. And there are going to be lots of applications like that. You've seen Lisa Project. That, of course, will be running on Mac. And we don't even know the kinds of applications that are going to come out in six months to a year. As an example, we'll be able to laser-print output from this thing by next June, and that is pretty exciting to us. So, if we sell these on a university campus, you'll be able to take your disk into the library and get output off a laser printer, which will be approaching typeset quality. That's the kind of stuff we're doing; you just can't do that on a current-generation personal computer.

And then the third thing is what Burrell and Larry and Andy and the other software people have done. When we shipped the Apple II, we fundamentally shipped about 2K bytes of ROM with system code. The IBM system's got 8K bytes, but it's really kind of loose as a goose; it's about 4K bytes by our standards of code. Mac has 64K bytes of the tightest, most elegant code that this company's ever written. Most of the computers now are basically shipping a file system and a few drives, but what's really interesting is that on top of that, we've layered on memory management and on top of this is Quickdraw.

Jobs: Mac's a completely open machine—we've got a book called *Inside Macintosh* that tells all the secrets of it. But we're going to try to get a little uniformity through the carrot rather than the stick. And the carrot is that there's a finite amount of RAM in this machine, and we've done all these things for you in ROM. Now, you can do them yourself, there's nothing that says you can't do them yourself, but if you do, you've got to write them, which is going to take time and means you're going to be slower to get to market; you've got to chew up precious RAM space, and the chances are pretty good that we did a better job than you'll do. So we're going to try through the carrot to get a little bit of uniformity in the user interface in some of the ways the things are done.

Hertzfeld: See, we're really a 192K-byte machine, and if the programmers want to throw away 64K, then they're doing a dumb thing.

Jobs: We're a 192K-byte machine that deep-freezes 64K.

Hertzfeld: Highly tuned, tested, debugged, highly compact, very fast, very high-quality consistent code.

BYTE: What are all the factors in this that make it go so fast?

Hertzfeld: Sweat.

Jobs: Burrell, Andy, Larry, Bill—how long did you work on Quickdraw?

Atkinson: Four years.

Hertzfeld: All of us care a lot about performance. Surprisingly, that's unusual. A lot of people don't care if their system's. . .

Atkinson: Like Quickdraw. I won't even count the first runs in Pascal, but the first runs in assembly language were running 160K bytes, before I added a lot of the new features. It's now down to 24K bytes with lots more stuff in it. Character-drawing speed is one you look at for drawing an arbitrary size character, an arbitrary starting pixel clipped to an arbitrary area. We were running, when it was being developed on Lisa, about 1000 characters per second the first time. Well, I got that up to 4000. Mac is running about 7000. That's seven times 9600 baud. This is typical of all of our software packages here. You go through, get the best algorithms first, get the stuff right. Then crunch it down, make a first pass in Pascal, get the algorithms right, find the cleanest algorithms, find all the corners, and make sure they're tested. Then I translate it into loose assembly language to get down into assembly language and get it working. Then I'll go through and get all the bugs out again, and I'll go through and do fine register allocation to figure out what's the most important thing. This little baby, the 68000, has sixteen 32-bit registers sitting there, and the way you get performance out of that is to keep them full. Keep the registers full of important stuff all the time. That's the way you make this processor sing. So you go down and you do register allocation, and then you don't stop. Then you feed it back, you get your people to use it.

'Optimization without measuring is wasted time. Find out where the application's really spending time and go whump on that code.'

Quickdraw was designed by "pull" from applications rather than "push" from the design team. You provide a facility, watch the applications group try to use it, understand where they misunderstood something—maybe you've got a bad model, you want to make it simpler and cleaner—or where they don't have enough performance. And then you go back and you measure, measure, measure, measure. Optimization without measuring is wasted time. Find out where the application's really spending time and go whump on that code. And any other cases they're very seldom using, squeeze them down in size, and stretch the other ones. There's always a trade-off between size and speed. Stretch out the common cases, let them be bigger and much faster, and then keep the generality by squeezing down the infrequent cases. So play your odds. People draw characters in OR mode a whole lot, and OR mode is about twice as fast as the other modes, so 95 percent of all characters are drawn in OR mode. Statistical measuring of the use of the thing allows you to get much more performance on your average throughput than you can if you don't go back and measure.

I think we all believe that system software should be done in assembly language at this stage of the game because high-level languages can't

give you the performance and the code density that you can get out of assembly language.

BYTE: So far, it has seemed that with all the systems that have mice, all those that are on the market, you pay a great price in terms of performance to get ease of use.

Atkinson: You make a responsive system; it isn't just draw some characters out there. It's also, remember where you put them because if the guy touches on them you want to light them up. There's a lot more guts in that application.

Jobs: It's not just systems that have mice. What's happening is there are a whole bunch of things that go with the mouse. It's not just hanging a mouse on a first-generation personal computer and using the same old, fixed-pitch text and things like that, just replacing four cursor keys. What we've done here is take a quantum leap, where, in addition to having the mouse be the major pointing device, we've gone to full proportionally spaced fonts, totally software-painted on the screen, any size, any shape... totally new architecture for displaying things to the user.

Atkinson: But the responsiveness is where the code goes.

Jobs: The responsiveness and the fact that there isn't a mouse-based system out yet that uses a 68000. We're obviously using the power of the 68000 in addition to this code.

Smith: There are some tricks we played in the hardware, too. For example, we knew that the ROMs would have real important things in them. So we made the ROMs sort of read-only cache memory, whereas the RAM has to contend with video and sound for access, so we cut that down to the bare bones, but the code that's in ROM, like Bill's graphics and the other stuff, can run as fast as you can run a 68000.

Jobs: If you look at the really great applications, even on first-generation personal computers, most of them are written in assembly language—Visicalc, 1-2-3—it's like if you're going to sell a million of something, it pays to handcraft it in assembly. If you're going to sell 10 of something, it probably doesn't. If we'd written this in Pascal, we would have been able to fit a fourth as much code in the ROM or would have to have four times the ROM, and you wouldn't have had the performance. Because we're going to sell 10 million of these things in the long run, it pays to superhandcraft it; we only have to do it once. Every time these ROMs are burned, it doesn't cost us any more engineering. . .it's all been done up front.

Capps: Because we cared enough to do it as well as we possibly could.

Jobs: We took a 12K-byte Pascal program running on a Lisa and we said we want to do this in 2K and make it faster. But we had that extra year to do that. And we also had the motivation, of course.

Atkinson: When you're writing assembly, you know each instruction is going to take 2 microseconds, it's going to take 4 bytes of memory. In Pascal, you're removed from that, so you don't concentrate on performance as much. When I'm doing I/O stuff in assembly language I look at

the theoretical maximum speed you can run at. Why not do it as fast as you can possibly do it? Especially when you're doing disk I/O stuff. How fast can you get into an interrupt and out?

BYTE: Andy, let's talk about the early days, after it had become Macintosh.

Hertzfeld: I don't know, there's something that makes a job a little more fun to work on when the odds are against you. And that's sort of how it was in the early days. I was maybe the fifth or sixth person to come work on it. Steve took me over to this little building separate from everywhere else, where there were these incredibly great people working on this little wire-wrap PC board. All it could do when you turned it on was write "hello" on the screen about 80 times. And everyone was incredibly excited to see it write "hello" on the screen because it meant that the central processing unit was there and all that potential was there to be mined. I spent my time mining that potential.

The very first time we got an early version of Quickdraw running, and we got the mouse going—that's just an incredible thrill. Or getting back the first PC board—we all went out for pizza on Friday night. We got the boards in about four o'clock Friday afternoon, and Steve said, "Well, if you get these done before midnight, we'll take you for pizza," and we stayed there...not because we wanted the pizza, but because we wanted to see that board working. And I think that none of our Mac PC boards have ever had to have a wire run to fix something, which is pretty amazing. That's the attention to detail that you just can't get people to do for money. We do it for love...this is the most important thing in our lives ...to make that great computer.

It's fun for me because I like operating on a systems program where I can operate in an environment where there's not that much support. In the early days when I first started here, the first thing I did was come in and write all kinds of crazy demos, stretching things around on the screen and making balls bounce, and one reason to do it was that I didn't want to write the system code until I was good at writing 68000 programs. So I just wanted to learn by having fun, and the other reason is that it gets people excited about it. Just this raw hardware sitting there doesn't do too much, but once you start making this fun thing happen and that fun thing happen, the excitement starts getting generated. You get to attract other good people, and one by one we picked up on more and more people. We were very, very selective; it was very hard to find people to work on Mac software, because on one hand we had the very high goals of doing this research, Xerox PARC-like stuff with uncommon, high technical standards. On the other hand, we had a very inexpensive, limited-memory machine. So all the Xerox PARC-type guys who came and interviewed said, "Oh, you don't have 2 megabytes? Forget it, I don't want to work on this thing." They're all used to their Dorados. But gradually we found great people like Larry and Bruce who were turned on by the dream, and they came and joined our band, and I guess we reached critical mass.

Atkinson: Most of the early people were recruited from Apple...and we have a pirate's flag that we sometimes put on the roof. The idea is we're pirates and we go around and try to steal the best we can from anywhere we can get it, and mostly that's been from Lisa. A lot of it's been from Lisa, but it's true in initially putting together the team, too; we try to get the best people we can from anywhere in the company.

Hertzfeld: One of the slogans Steve came up with when we had a retreat in January was "Let's be pirates," the idea being that we were mavericks out to blow people's minds and overturn standards, create new standards, not do things like everyone else.

Atkinson: There was always the thrill that this was going to be the one project that was probably the most amazing thing you were going to be doing in your life.

Hertzfeld: And the other slogan was "The journey is the reward."■