RISCy Business

Description: After catching up from a very busy week of security news, Steve recounts the history of the development of complex instruction set (CISC) computers following their evolution into reduced instruction set (RISC) computers.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-252.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-252-lq.mp3

Leo Laporte: It's time for Security Now! with Steve Gibson, Episode 252, recorded June 10, 2010: RISCy Business.

It's time for Security Now!, the show that covers everything you, you, yes, you need to know about security. And fortunately we've got a great guy to do it, Mr. Steve Gibson. He is the guru at GRC.com, Guru Research Corporation. No, Gibson Research Corporation. He is also the guy who discovered and coined the term "spyware," wrote the first antispyware program, has written many free security tools, all of which are available at GRC.com, and the creator of SpinRite, which is his bread and butter and the best hard drive maintenance utility out there to this date. Although, Steve, you're going to have to do something when everybody goes solid-state. I don't know what you're going to do.

Steve Gibson: That's true.

Leo: CryptoLink, maybe.

Steve: CryptoLink to the rescue, yes.

Leo: Yes, that's the next stage.

Steve: People are worried it's never going to happen, well, now it's going to happen.

Leo: It has to happen.

Steve: It's got to happen.

Leo: Or Steve is out on the bench there, the park bench, feeding the pigeons.

Steve: Though I guess the good news is that SpinRite tends to be used on drives people have had for a while, which have finally given up the ghost, and it gives them another life. As opposed to, I mean, although a lot of people use it on drives they buy fresh because they want to check it before they stick it into a machine, just like you guys do. So it certainly is useful when your drive is coming out the box. So it'll have a long tail on it. But it is not the case that it helps solid-state drives at all, in any way. And it would be bad for them. So don't run it on SSDs. That just doesn't make any sense.

Leo: What is the topic of the day today?

Steve: We have - there's a couple more things I want to finish up on our sort of fundamentals. And the title for today's show is "RISCy Business," where RISCy is R-I-S-C, in capitals, being an acronym for Reduced Instruction Set Computing." I want to talk about - and this is going to be, I think, really fun - the evolution of the architectures of computing from the architecture we've described so far, sort of a basic, starting, simple, this is what instructions look like and how the computer works. Today I'm going to take us all the way through this revolution in the way computers are designed into sort of what happened with them as they got increasingly complex, what the pressures were from various sides, and what the result was. I think people are going to find it very interesting. And we've got a ton, boy, it's been a busy week in security activity.

Leo: It's true. We moved the show a little bit from Wednesday to Thursday. That might be a good thing because more stuff has happened since the show - we usually record this Wednesday at 2:00 p.m. Eastern. And because Paul Thurrott's schedule didn't accommodate that, you flipped with Paul. Thank you...

Steve: Yes.

Leo: ...for doing that. So and it turns out this was probably a good thing because there's even more stuff to talk about. So what should we start with?

Steve: Well, we'll do updates first. There wasn't much, but what there was is important. The big news was that Adobe got hit with the news of a new, previously unknown, zero-day exploit which was discovered in the wild Friday. They were informed a little after 10:00 a.m. Pacific Time almost a week ago. And they acknowledged the problem. They posted some news on their site. I blogged about it. And so anyone who had subscribed to the steve.grc.com blog would have found out about it at that point.

In fact you and I talked about it on your Tech Guy show on Sunday because it was regarded as a very critical vulnerability. It affected Flash, so going to a site where you had Flash active could allow some malicious code to get into your computer and take over. But also Adobe Reader and Acrobat both have Flash components. That is, you are able to put Flash in a PDF. And that component was vulnerable, as well.

So the remediation of this, dealing with this problem, was of great concern for people. Adobe had previously taken about two weeks to respond to something like this. The good news is they've cut that in half. They've said that today, as we're recording this, Thursday, June 10th, that at some point today they will have a fix for the Flash portion of this, but not for the PDF and Acrobat portions for another two weeks, not until I think they said June 29th. So the largest exposure they're going to be dealing with.

On my blog page, which is the current blog, steve.grc.com, I've got links to Adobe Labs to deal with the Flash problem. So people who had responded immediately, what I was recommending people do is jump ahead, rather than waiting for Adobe to fix their v10.0 point whatever it is Flash - I think it's actually .45.2 - to jump ahead to 10.1 because Adobe originally said that was not vulnerable. They believed it was not vulnerable. Now they've confirmed it's not vulnerable. And it's at release candidate 7 level. It's very stable and reliable. Lots of people are using it. So all of this week people who had received that information from me were able to protect themselves.

And I'll remind our listeners that they can use the multi-browser Mozilla plug-in check: Mozilla.com/plugincheck will, on any browser, check to make sure that the Flash plug-in is the most recent. At this moment it says everything's fine because what it's saying is that your plug-in is current, even though what's current is a problem. So what will happen, if you do it again later today or tomorrow, Friday, after Adobe has published their update, is then your plug-in check will say, oh, you no longer have the most recent one, click here to update.

So that's an easy way for people to check when Adobe has released the official fix for the 10.0 version of Flash. If listeners want to do something immediately, I would say without hesitation go to labs.adobe.com, and you can install instead the next major release, which is just about ready for release, but not quite yet, which is v10.1, and you'll be okay.

Leo: So the Firefox plug-in doesn't tell you that you're insecure. It only tells you that there's an update.

Steve: You're current, right.

Leo: So this is important to understand. Because I think it kind of gives you the presumption that, oh, it's checking to see if I'm insecure. It doesn't do that. It only says there's a new version if there's a new version.

Steve: Correct.

Leo: You need a new version, but there isn't a new version, so it won't tell you.

Steve: Right. And if it...

Leo: And it won't tell you about betas, apparently, either.

Steve: No. And if it told you you were vulnerable, well, you would then say, okay, what do I do?

Leo: Now what, right.

Steve: And it's like, well, they don't have anything for you to do because Adobe hasn't released a fix yet.

Leo: Right.

Steve: Now, this only handles, unfortunately, for the next two weeks this only handles, no matter what you do, whether you wait for Adobe to release a fix for 10.0, or you jump ahead to 10.1 as I would recommend, this only fixes the Flash side. There's a problem with PDFs. And we know that the bad guys are probably going to recognize that only part of this has been fixed, for whatever reason, and may start targeting people with PDFs. There is a file called - boy, I'm blanking on it. It's auth something. Well, I blogged it. It's steve.grc.com again, my blog. There are two things you need to do. You need to deal with the Flash problem, and also with the Reader problem.

Leo: I stopped using Reader a long time ago because of all these issues.

Steve: Right.

Leo: I use Foxit, but...

Steve: And so people who do have Reader as their registered...

Leo: So it's authplay, a-u-t-h-p-l-a-y dot dll.

Steve: Dot dll, right, authplay. And so the recommendation is to find instances of that on your machine - and I provide the path where it'll be installed with Reader and/or Acrobat - and just rename it to, like, authplay.xxx, so that your system won't know it exists. It won't be...

Leo: What will happen? Will it break the Reader, or...

Steve: Yes. If you then clicked on - if your PDF opened and tried to invoke the Flash, you'd get an error saying...

Leo: Oh, Flash won't work. But the Reader will continue to work.

Steve: Precisely.

Leo: Got it.

Steve: Exactly. So only a probably malicious PDF wouldn't work, and that's what you want is for it not to work. So basically you would be neutering the PDF's ability, the Reader's ability to play Flash, which is, I mean, whoever wanted a PDF to have Flash content in the first place? I was like, okay. And apparently you're able to disable that feature using the control panel, but it still doesn't protect you from this vulnerability.

Leo: Okay.

Steve: Go figure. So anyway, I guess I would recommend, given than we now know Adobe has formally said they're not going to have a fix for this problem with the PDF aspect of the vulnerability for another two weeks, I would be uncomfortable knowing that, if I opened a PDF, it could get me. So I would follow the recommendation of renaming authplay.dll to, like, .xxx, and then you're going to be safe until Adobe fixes this. And when you install an update to that, it'll just put in a new authplay.dll, and your old .xxx will - you could delete it at that point. I mean, you could delete it now, for all anyone cares, probably. Probably safer just to rename it, though.

Leo: Yeah, okay, cool.

Steve: For once, Windows Patch Tuesday is not the top of the list because it's just sort of another one, okay, fine. We've got 10 security bulletins. Three out of the 10 are maximum rating of critical. This eliminates 34 new known vulnerabilities in both Windows desktop and server OSes in Internet Explorer and Office. So we are now passed the second Tuesday of June. Microsoft has released these. Of course the advice to everyone is update yourself sooner rather than later. And you probably have to reboot your system, so choose a comfortable time to do that.

And I did note, although I don't think this is a big problem, just because it doesn't represent a large attack surface, Adobe Photoshop - once again Adobe - both CS, CS2, CS3, and CS4 have a known critical vulnerability in them such that, if you opened an image in Adobe Photoshop that had been maliciously crafted, you would be in trouble. Like I said, well, okay, that doesn't seem like a huge problem. But I just wanted to let everyone who does have those versions of Photoshop to go check now, updates are available, so just ask Photoshop to check for updates for itself. And if there is one, that's what you want.

Leo: All right. We've done the updates, what's out there. Now let's get to security news because there's some big stuff going on.

Steve: Yeah. Probably the most alarming story broke very recently, relative to our recording of the podcast, and that was the news which, unfortunately, was overwrought, I would say is probably the right term.

Leo: Mm-hmm.

Steve: And also blaming the wrong person, in my opinion. The news was originally posted by Gawker.

Leo: Not known for its security coverage.

Steve: No. And their headline, even now, after it's clear that this is not the case, says that it's a huge breach in Apple's iPad security, naming Apple as, like, the focus of this. So here's the story. A group of researchers at a company called Goatse Security, security.goatse.fr, they discovered the protocol which AT&T was using to fill in the email address field for the log-in to their system on the iPad. So the idea would be, an iPad user who wanted to check on their AT&T account status would bring up the control panel in the iPad, and the email address is one of the two authentication fields which you fill in. So what the iPad was doing was it was sending what's known as the IDD, I'm sorry, the ICC-ID, to AT&T. That stands for Integrated Circuit Card Identifier, which is part of the SIM, the standard SIM card. SIM stands for Subscriber Identity Module. So that's part of the standard data in the SIM card is this ICC-ID. So that was going on the fly to AT&T's servers and essentially making a standard web request, an HTTP request from the iPad to AT&T's servers' database. The servers were then responding with the email address of the user so that it could...

Leo: Oh. In the clear.

Steve: In the clear, so that it could populate that field. Which made it - it was a convenience feature which made it easier for then the user to just - all they had to then provide was their password.

Leo: Right.

Steve: That matched their account name, the account name being this email address that they had used. So the bad news is that there was no security protecting this. This was an in-the-clear standard HTTP query and response, like we've talked about on the show endlessly. So the Goatse guys realized that anyone could make such a query of AT&T's backend database of subscribers' email addresses, using made-up ICC-IDs. The ICC-ID is a fixed-format international standard, part of GSM, which goes along with the SIM cards. It's 20 digits long, the 20th digit being a check digit. So it's sort of a checksum for the 19 which precede. The 19 digits are fixed fields, sort of like a MAC address, of known fields, like for example the carrier's identity and other stuff. And then there's a chunk of digits at the end which is like a serial number. So many excited iPad owners were taking screenshots of this panel of theirs, because this, you can see, you bring up sort of like in the About dialogue on an iPad, and it shows you your ICC-ID, which is this 20-digit number. So there were, you know, many people were sharing this without any concern on the 'Net. And frankly, any iPad owner could easily look at it.

So what the hackers did, cleverly, was they reverse-engineered the protocol, which was trivial because you only had to, in the user agent field, you had to pretend to be an iPad.

So you used the iPad's user agent to make it look like an iPad was making the query to AT&T's servers. And then they just set up a PHP script to try all possible ICC-IDs within the range that were known to be iPads because they're sequential, unfortunately. So they started, like, at ICC-IDs close to those that had been shown publicly, and they just had their script try them all. And what was embarrassing to AT&T and of great concern to many people is they collected 114,000 email addresses, like many, like, .gov and army.mil. And apparently, you know, A-list celebrities and all kinds of government officials who were, when you see their email address, you know who it is because, you know...

Leo: Right, like Rahm Emanuel, chief of staff.

Steve: In the White House, exactly.

Leo: In the White House, yeah.

Steve: And so the news that broke was that this was a - of course unfortunately Apple was dragged into this because they made the iPad, although this was entirely AT&T's fumbling of not protecting the email addresses of their 3G customers better. It would have been certainly possible, I mean, I would argue, unfortunately, that it's probably not ultimately protectable because, as we know, if the iPad could generate a query, then it's possible no matter what to reverse the iPad's generation of the query and pretend to be an iPad. So, I mean, but AT&T could have made it so much more difficult, could have raised the bar so much that it would have never been a problem.

Leo: Well, they could have hashed it or something. I guess they can't, though, because the software...

Steve: They would have had to encrypt it because they couldn't hash it because they want to show you what your email address is. So they would have had to have done reversible encryption in order...

Leo: So that capability would have had to have been built into the software.

Steve: Precisely. And so if the software were reverse-engineered, somebody could issue their own. They could just arrange to, like, change the ICC-ID and have the iPad do it. So the bar couldn't have been raised all the way up. But it could have been so much higher that, well, I'm sure that these guys were doing some packet sniffing, and they saw the data in the clear.

Leo: Right.

Steve: And they said, hey, wait a minute. But if instead it had just been scrambled, it would have never occurred to them what was going on. So, yes, AT&T blundered by not protecting this data more strongly. And that's...

Leo: Now, what did they get, though? They only got an email address.

Steve: Yes.

Leo: How, well, let me ask you a question. I mean, look, I just showed my ICC-ID. Now, this, by the way, was patched by AT&T, fixed on Tuesday.

Steve: Prior to the release of the news. So there was some responsible disclosure done so that, prior to the release, AT&T was allowed to fix it so that it wouldn't still exist.

Leo: And Goatse, by the way, tried to give this story to a lot of mainstream media, all of which ignored it, probably because of the name Goatse Security. I would have ignored it. It sounds like a 4chan prank. But how bad is it, really, if they got, I mean, I show my ICC-ID. Even if that hack still was out there, people would then have my email address, which is available publicly in many, many places. Is it - how bad is it that your email address gets out?

Steve: Well, I think that's for everyone to judge.

Leo: Right.

Steve: Probably the concern is...

Leo: That's all they got, right, is email addresses.

Steve: That's all they got. They got the email address that was used for the 3G account. I made up an email address for the purpose. There's some - I didn't use my real email address just because that's who I am. But so people have oftentimes multiple email addresses. They may have scratch ones or discardable ones or who knows what. So the concern is, yes, I mean, I guess I would say it's controversial. You could decide it's a big thing; you could decide it's not a big thing. But you're right, Leo, all they got was a large database of early adopters' of iPads email addresses. Would it have been better if they had not gotten that? Absolutely.

Leo: And maybe some embarrassment from the people with the mil addresses that they were using their business, corporate, government address. I would hope Rahm Emanuel wasn't using his super-secret White House address for registering his iPad. I mean, that would be kind of dumb.

Steve: Well, unfortunately, we know that dumb stuff happens all the time, which actually gives us plenty to talk about on the show.

Leo: And I won't belabor it, but we were trying to decide - the story broke while Tom was doing TNT, our new daily news show that he does at 2:30 Pacific, 5:30 Eastern every day, Monday through Friday on TWiT Live. And Tom and Becky Worley were in here. And Dane came in with it, he saw it and said, "They might want to know this." And we were afterwards debating whether - how to cover it. And I think what Tom did was good, which was he said, "This is just coming across. We'll look into it more." Certainly it's not as sensationalistic as - and I think Gawker is probably not well prepared to break a story like this, frankly. So we decided, we opted not to cover it as a big breaking story. And now I'm glad we didn't. I mean, that's kind of the issue, is how do you cover something like that?

Steve: Yes. And it does take some time to process this. I missed a call from Reuters yesterday evening. They were trying to get a hold of me for exactly that purpose, to help them understand...

Leo: How big a deal this is.

Steve: ...if this was a big deal or not. And I would have said to them, eh, you know, here's what it is. I can't make that judgment. That's a value judgment. But I can tell you the facts. The facts are, due to a mistake on AT&T's part, a big block of email addresses were sucked out of AT&T's database. They should not have allowed that to happen. The individuals who subscribed can change their email addresses. They can be more the wiser now. I mean, there's enough spam in the world. I would imagine lots of people have these people's email addresses already. So one more person does.

Leo: Yeah. Good. Well, there's the story. And that's why we do this show, so you can get an intelligent, clear explanation of what really happened.

Steve: So the never-ending tale of Google's WiFi inadvertent plaintext capturing.

Leo: Yeah. Which is another case of where others have been fairly sensationalistic, and I think you've been very level-headed.

Steve: Well, yes. Okay. So this is the story that refuses to die, although certainly Google wishes that it would. Canada has now joined the fray, adding themselves to Germany, Italy, France, and of course the U.S. FTC, who we talked about last week "investigating," unquote, this. What happened was, I mean, the news of the week is that Google hired an independent third party to analyze what they did, what happened. And the third party produced a report, I don't remember how many pages now, like 20-page report, which analyzes the source code that Google used as part of this.

And this is another one of these classic controversies where, if you want to read it as bad, you can; and if you want to read it as not so bad, you can read it that way, too. Because the source code contains a bunch of defaults for the way the WiFi sniffing will work such that it can be configured not to save encrypted data. And the sniffing software defaults to not saving the packet payload, the contents of the WiFi frames, as they're called, because really the only thing that Google cared about, I've always asserted, was

the header information which contained the SSID and the MAC address. That's the information that was valuable. And it seems entirely reasonable that they use sort of generic software to obtain that.

And unfortunately this generic software had a default so that it would save the payload of the packets when it was not encrypted. And so in this report they show the bit in the packet which identifies whether the payload of the packet is encrypted or not and that, in the case that encryption is in use, the payload is not saved. So what's a bit surprising, and people who think Google did wrong on purpose, they could say, aha, Google wasn't saving everything, they were only saving the things that they could read. On the other hand, you could say, well, but the software that they used unwittingly knew better than to save stuff...

Leo: Garbage, right.

Steve: ...that was encrypted because you know that's pseudorandom noise.

Leo: Somebody in the chatroom is saying they were using an off-the-shelf program called Kismet. Is that the case?

Steve: Yes, yes.

Leo: Oh. Well, so they were just using - oh.

Steve: Yeah. I mean, it was literally this, and then they wrapped it in some of their own code, all beginning with G's - G this and G that, short for Google, obviously. And so maybe their case, maybe the "we weren't using any of this" would have been stronger if they'd been saving everything. Everybody wishes they'd been saving nothing. What they were saving was the stuff they could read. But I don't think that...

Leo: But not because they wanted to read it.

Steve: Precisely. And remember that I guessed a couple weeks ago that these vans that were driving around, and I guess they have people on fancy bicycles, also, that they were just streaming this all in and doing no analysis on the fly. Turns out that is the case. I was assuming that they were just out roaming around, sucking all this in, adding the moment-to-moment signal strength, which actually is part of what Kismet records, and the GPS metadata so they would know where they were when they recorded this, and just stuffing it all on hard drives to then be analyzed at leisure offline, outside of the vehicle that was doing this roaming, which is exactly the way the system worked. So that's the story. To me this provides additional detail. But what I'm seeing is that people are jumping on this, saying, oh, look, this report further incriminates Google. And I don't read it that way. But again, it's understandable how somebody who's really determined to do so, could.

Now, I noticed in other news the California Ninth Circuit Court of Appeals just upheld a ruling by a lower court - and this is in an entirely unrelated case, but it relates to this

issue - denying damages in a class-action suit brought by some random guy on behalf of a bunch of other presumably damaged people whose Social Security number was contained in a laptop that was lost. I think it was, shoot, I can't think of the retailer now. It's a clothing retailer, short name. Anyway, it was a clothing retailer who had...

Leo: Kohl's, Kroger, Macy's, they're all short.

Steve: Gap, I think it was.

Leo: Gap, that's even shorter.

Steve: Yup, I think it was Gap. And so somehow he'd, you know, his personally identifiable information - they, in responsible disclosure, let him know that a subcontractor of theirs, like Venture or somebody, had this laptop, and it got lost. So he's upset and sues because he's annoyed. And so what the lower court ruling asserted through a careful reading of the California Constitution was that actual concrete damages, proven damages must have resulted, and that simply being annoyed is insufficient.

Leo: Thank you.

Steve: Yes.

Leo: You can't sue for being annoyed.

Steve: Right. Right. And so here again this, of course, bears on the Google issue because there are now several class-action suits that have been filed against Google because people are annoyed. And it's like, well, okay. I don't know who's going to, you know, only the attorneys make money on this. So to me this is a non-issue. But the good news is maybe the word will get out that this is not any way to cash in on a mistake, I believe an honest mistake, that Google made. Certainly they would do it differently if they could.

And my last little bit of security news is just an update: Windows 7 Service Pack 1 is due out around the end of July. So a little less than two months from now. Anyone installing new versions of Windows 7 won't have to go through the laborious process of installing all bazillion security updates which have accumulated since the release of 7. You'll be able to install the Service Pack 1 and catch up to be current at that point...

Leo: Excellent.

Steve: ..in the end of July. In errata, I just wanted to mention a couple things. Several of our listeners wrote in, Leo, to tell us what "rooter" meant.

Leo: I know what "rooter" means. I know, I know.

Steve: And I didn't realize...

Leo: In only, well, in Australia.

Steve: In Australia, apparently, maybe it means horn dog. I'm not sure.

Leo: Not exactly. Rooting is like rutting. In the states we use the word "rutting" in the same exact context.

Steve: I see.

Leo: Not to make this show not safe for work, but that's what's going on. So in Australia they pronounce it "router." In Britain, where the connotation doesn't exist, they pronounce it "rooter." I don't care. You call it whatever you want.

Steve: I don't, well, router? We've agreed.

Leo: We've agreed.

Steve: Router is what it is because...

Leo: We have a term here.

Steve: ...we're firmly lodged in California. I do want to mention, for all of the people who own Kindles, not to feel badly. I prefer it over the iPad to reading. It's just I've...

Leo: That was one of the questions, you know, the jury was out. Now, is it because of the screen for you?

Steve: It's everything. It's the screen, it's the weight, it's, I mean, clearly Apple is aware that there's a glare problem with an LCD. It can easily be glary because normally you're reading black text on white. And the Kindle software has an independent brightness control so that you're able to dim the screen below where it's normally set for when you're reading.

Leo: And I use that in the darkness.

Steve: Yes.

Leo: Which, by the way, the Apple software also has.

Steve: And so clearly, if the Apple software has it, too, people are sort of aware that there's a glare problem. When I used to be reading on my Palm Pilots, or my Palm - after a time they were Pilots - I was reading white text on a black background that was much more pleasant for me to read. So I actually do find that a reflective screen is easier on my eyes. That is, the original eInk Kindle screen, somehow it's just - it's taking the photons that are available in the environment and bouncing them off the screen rather than emitting any of its own. And also the size, the weight, I can't really say the battery life because it's just not an issue on the iPad, the battery life is long enough thanks to the ARM-based processor, which we'll be talking about by the end of this episode, how and why it lasts so long and gets such good battery life. So I just wanted to mention that. In fact, it's "Where Wizards Stay Up Late," the book you recommended last week. I got it.

Leo: Oh, good. How do you like it?

Steve: I've begun reading it. It's fun. I don't think it has a huge audience. That is...

Leo: No. That's why it didn't sell that well.

Steve: Yes. I wouldn't expect it would. I mean, I'm enjoying it because I like the history of all this, and I lived through it. And so it's like, oh, now I know exactly where the word "packet" came from, which I didn't know before.

Leo: Yeah, right.

Steve: But it's like, okay, I'll just tell everybody where it came from when we discuss it here in a couple weeks. And they don't have to read the book to find out. So but I'm enjoying it. But I was noticing some discomfort on the iPad. It was just maybe - one of the things is there's just so much text there. Maybe I've gotten used to more bite-sized pages, sort of like what the Kindle provides. I feel like this huge page of text on the iPad, like the screen's too big. And so in fact I went all the way and installed the Amazon Kindle Reader on my iPod Touch and tried reading some more of it Sunday. And I decided, okay, that's too small. So the Kindle sort of seems to be just right for me. So I'm not unhappy that I have it.

Leo: I wonder if people who - see, I read mostly in bed. So I like a device that is backlit because I don't have to turn on a light, doesn't bother Jennifer. And so - and I dim it because it will hurt my eyes. But when it's dimmed, in fact, the Kindle has a sepia color that I like a lot.

Steve: Yes.

Leo: I find it easy to use, legible. And the truth is I don't hold the Kindle or the iPad in the air. In both cases they're resting on something because I'm not going to...

Steve: You probably - maybe your stomach, Leo?

Leo: My stomach or a pillow. Sometimes I read sideways. And even the Kindle, it's not the weight of the Kindle, it's your arms. It's the weight of your arms that's the pain in the butt. So if you were, on the other hand, reading anywhere where there was bright light, outside particularly, the Kindle would be a clear choice. So I think depending on - but it's interesting. My wife prefers the Kindle. My mom has now an iPad and a Kindle. She prefers the Kindle. So I think it is probably a little easier on the eyes. For the occasional reading that I do, and programming manuals, things like that, I actually like the iPad and use it. I gave up my Kindle. My wife has it now.

Steve: And I have to say, the fact that synchronization works so well...

Leo: It's nice, yeah.

Steve: And I guess Apple announced that they would have...

Leo: They're going to do it, too.

Steve: ...cross iBook synchronization, too. So that's really wonderful. I mean, you can grab the device that's best for your current situation.

Leo: Exactly. I mean, not everybody's going to have both. But if you can afford to have both, it is nice to have that capability. And I do in fact do that all the time.

Steve: Yeah.

Leo: And that's because it's also on the iPhone, and so I can - and the new iPhone, with that screen, might be a very - I have to see it, but might be a very good reading device. I'll be very curious if it's a good reading device.

Steve: I'm very interested in the screen, too. I did note that a physicist took issue with what Steve claimed, the so-called "retina display." I'll fill in our listeners a little bit. I mean, I'm very screen-oriented. I hail from the light pen on the Apple II, which is one of the products that I designed in a previous life. I'm extremely resolution sensitive. I mean, I think the more is better. And so whereas the current generation iPhone and iPod Touch are 480x360 resolution - yes, or is it 320? 320, 480x320. The next-generation iPhone doubles the resolution in each direction. So you have four times the number of pixels total. It's 960x640. And Steve Jobs was claiming that, at about 12 inches away, that the resulting 326 pixels per inch was below the eye's ability to resolve pixels.

Turns out that's not the case. The way to think of it, the retina's resolution is 50 Hz per degree, that is, 50 cycles per degree. And so if you calculate what that means at 12 inches, that actually means about 477 pixels per inch is the retina's ability to resolve. Which is not to take anything away from Steve and his retina display, and certainly the spectacular resolution of the screen. But the person who did the analysis said, let's try to keep people honest.

Leo: Let's be honest, yeah. It was actually, the one I saw was by the guy who heads DisplayMate, which is a company that makes software for calibrating monitors and has been in this business for years and years and years. So maybe this physicist did the same thing. Now, if you hold it at arm's length you can't resolve pixels. But the point is, nobody's holding it at arm's length.

Steve: Exactly.

Leo: My issue is, when you get dot densities that high, sometimes I find it, as an older guy, harder to read because the icons and text get smaller. Yeah, they're clearer, but they're too small for me to read. So I'm reserving judgment until I hold one of these and use it for a while.

Steve: Yes. One of the problems is that - and this is something that I do trust Apple to do, and I see other companies get it wrong. And that is, for example, when I talked earlier about reading books with the screen inverted on my Palm, when I was looking at white text on a black background, it was very often the case that it would kind of get pinched, that is, it was a serif font - unless I manually overrode the font, which is what I ended up having to do in order to get a stronger, non-serif font to have it look right when it was inverted. So things like the way pixels will bloom a little bit, if you invert them, then what was blooming before becomes pinched, and it just doesn't look right. But again, that's the kind of detail that Apple really does - is good at taking care of.

Leo: Right, right.

Steve: So I have a feeling they'll do the right thing.

Leo: Be interesting to see, yeah.

Steve: I have a fun SpinRite story to share with our listeners. This is a "SpinRite Saves the Wedding." From Darren Bessett in Thornton, Colorado. He wrote, "Dear Steve, here's yet another SpinRite success story to add to your collection. A good friend of mine and teaching colleague, is getting married later this month, in June, and has been preparing for an elaborate wedding of over 200 guests. Like most couples planning a big wedding, she and her fianc had developed a detailed wedding plan to manage the event, including numerous electronic files such as an invitation database, digital photographs, and PDF event contracts."

Leo: Geez, Louise.

Steve: "All of this data was stored on a laptop computer which, as you may have already guessed, is where the story is going. It one day failed to boot because of a faulty OS hard drive. And after repeated unsuccessful attempts to get the machine started, the couple had given up all hope. They were absolutely devastated because the bulk of their wedding plans were now trapped in a laptop that would not boot. How could such meaningful and important information succumb to a cheap electronic device?

"Enter SpinRite into the story. Upon hearing their dilemma, I immediately thought of the myriad stories you have chronicled over the years detailing the amazing feats of SpinRite. Although I've been using SpinRite for five years as a maintenance utility, I never needed to fix a faulty drive." Ah, guess why? Because he's using it as a maintenance utility. "Here was my opportunity to give it a try for that purpose. After configuring the broken laptop's BIOS to boot from the CD drive, I ran SpinRite Level 2. And within an hour the defective sector on the drive had been identified, and data recovery was underway. Afterwards the laptop booted, and the wedding files were all retrieved. Wow. Needless to say, the couple was overjoyed with the result. Although probably not the most romantic wedding gift, I will be giving them their own copy of SpinRite. Thanks again, Steve, for a great podcast that truly works, and for saving the wedding day." Or, I'm sorry, "for a great product that truly works" - the podcast works, too - "and for saving the wedding day. Sincerely, Darren Bessett."

Leo: Hope he gave them something else in addition to SpinRite for their wedding gift.

Steve: Hey, they may have been happy, given the fact that they wouldn't have had a wedding otherwise.

Leo: Hey, true. I think the gift was he got their computer working again.

Steve: Exactly.

Leo: That's the gift. All right, Steve. I'm ready. I've got my thinking cap on. Some may call this a dunce cap, but it's a thinking cap. And I'm prepared to think. Tell me, tell me, sir, how do we - where do we go now that we've been building a computer?

Steve: We've established a beautiful foundation over the series that we've covered so far, the idea being, of course, to demystify what a computer is, how it works, what it does. We know that there's a main memory, and there's a program counter that points to a current address of a word in memory, and that that word is composed of a bunch of bits, and that those bits are broken into fields, the most important field being the so-called "opcode," the operation code. The pattern of bits in there describes to the computer or specifies what the computer should do with that instruction, each one of these being an instruction. And the rest of the bits in this word provide the parameters for the instruction. For example, it might be "clear the accumulator." Or it might be "load from a location in memory into a register," or "store the contents of a register into a

location in memory."

So, and we've looked at what the stack does, the notion of the convenience of being able to have sort of a scratchpad where we can push things on, and we can pop them off in the reverse order so that as long as we kind of keep track of our own housekeeping, we don't have to pre-set aside areas to read and write into. Everybody's able to share this as long as they do so carefully. And so we've talked about subroutines. We talked about hardware interrupts and how interrupts from physical I/O devices which are much slower than the computer is running very fast, are able to be used to yank control away, yank the program counter to somewhere else, allowing the computer to service this interruption, send characters out to a slow printer, for example, and then return to the code that was running as if nothing had happened.

So we've got this architecture, and that's where everything began. What I want to do today is describe the evolution from there to now. That is, what happened after this and what were the pressures that were on the industry and on the engineers and on the companies that evolved this from that very clear and sort of basic beginning to present-day machines. So one of the things that happened, as hardware became less expensive, and programmers were complaining to the engineering team that, you know, like hey, be nice to have some more registers here. We've got the accumulator, but we're having to use it sort of as a scratchpad for main memory. We can't really hold much of what's going on in the accumulator. So they said we need some more - we need more registers, more accumulators.

And so the engineers said, okay, well, that means we're going to have to make the word longer because we're going to need some more bits in the instruction to specify which register you want to load and store and add and so forth. And so if the engineer says yeah, okay, fine, do whatever you - or the programmer said that, oh, that'll be fine. So words got longer. And then some of the engineering guys said, well, you know, a simple instruction like "clear a register," that doesn't have a memory address. It doesn't need a memory address. So that one could be shorter. And how about, the engineer said to the programmers, what about if, like, you could add two regions of memory and store it in a third. And the programmer said, oh, you mean like with one instruction? And the engineering guy said, yeah, why not? And then the programmer said, well, that would be fantastic.

Well, of course that would mean that the instruction, a single instruction, would need to contain three memory addresses for an add. It'd have to be, you know, the address of one of the operands, the address of the other operand to be added, and the address of the location where the result would be stored. Well, memory addresses take lots of bits in order to address a region, you know, enough memory. So this kind of an instruction would be really long.

So what evolved was, and this was a major break, was the notion of variable-length instructions. We started off with fixed-length instructions on our very simple machine, where most of the instructions were memory reference, and only referencing one piece of memory at a time - load from this memory, store to this memory, add from this memory to the accumulator, or/and from the memory to the accumulator as we've discussed. But as machines became more powerful, essentially because hardware prices were coming down, integrated circuits were happening, it was possible, it was practical to put more complexity into the systems. The programmers were demanding more features.

And so if you were going to have an instruction which had three memory addresses in it, it was going to be really long, multiple bytes of memory. But if you had an instruction that was going to just say "clear an accumulator," that could be one or two bytes. So we

broke from this notion of fixed-length instructions into variable-length instructions. And at that point all bets were off because now we could make our computers very complex. They could have, for example, you could have an instruction that was a prefix byte that said, you know that opcode we had, well, here's a whole 'nother set of opcodes. So now we weren't limited to eight or nine or 12 opcodes. We could have hundreds. It was practical to have many more possible operations.

So the engineers said, okay, well, what are we going to do with all this power? And the programmers said, well, since you're asking, there's some other instructions we'd like to have, like how about - we do a lot of things with linked lists, so how about an instruction to do linked lists? And the engineers said, wow, that'll take, like, a lot of manipulation. And the programmers said, well, what are you guys doing? Go do that for us. And so the engineers said, okay, write that down, Harvey. We've got to go implement that.

And then they said to the programmers, what else do you want? And they said, well, we do a lot of subroutine calling. And we know that when we call a subroutine we need to save all the registers. But it's tedious to have to, like, push this register and push that one, then push the next one, and push the one after that, and then before we exit to pop them in reverse sequence. How about an instruction for calling subroutines where, like, we had bits in the instruction which specified which of the registers to preserve in the subroutine call. And the engineers said, oh, we like that, that's cool. Write that one down.

So the programmers got very creative with the stuff they wanted. But then the engineers who left that imaginary meeting went back to their own area and said, okay, now we're in trouble. How in the world are we going to arrange to implement instructions this complicated? I mean, these things had sort of gotten out of control. And remember that the very simple fixed-length instruction computers were, for example, in the case of a PDP-8, they were contained on just three 8x10 circuit boards that weren't very dense, that just had simple AND and OR gates on them because everything happened in a single cycle.

Well, that's one of the other things that changed. As these instructions got more complex, no way were you able, for example, in the case of pushing a random, well, not a random, a specified subset of registers, well, this instruction was going to take, could take a long time to execute, one instruction, because the instruction would specify some number of registers get pushed. So it's going to have all these memory cycles after it reads the instruction, all these memory cycles to push these registers in succession. And there's going to be a reciprocal instruction to pop some subset back off. So that's going to take many cycles. And an instruction for managing a linked list, there actually was such a thing in the VAX. The DEC VAX had a linked list.

Leo: Really. That's such a...

Steve: Oh, yeah.

Leo: ...high-level primitive, I mean, golly.

Steve: Yeah, it was amazing how complicated these instructions got. There were some that were like, the programmers said, well, you know, we spend a lot of time trying to find the first bit which is set in a long run of memory. We want to, like, find a bit. How

about an instruction that just reads memory until it finds a bit set?

Leo: Oh, geez.

Steve: I mean, and there is such a thing. There is that instruction. And so, for example, in the famous Intel x86 instruction set, instructions can range from one byte long to 17 bytes long.

Leo: Wow.

Steve: There's that kind of range. So the engineers who actually had to create a machine that would deliver this kind of power said, okay, look. We cannot...

Leo: Give us a break.

Steve: Yeah. We cannot design hardware that, I mean, do you know how many AND and OR gates it would take to do this?

Leo: We're not going to do your job for you, for crying out loud.

Steve: So what they said was - I mean, and it was probably an instruction like this pop multiple registers thing. They said, well, now that we've got all these cycles that it's going to take, we need microcycles. We need something like a computer in the computer, which can be smart enough to implement very complex instructions. And so someone said, what are we going to call that? Well, we'll call it "microcode." And it's like, oh, okay. So what they did was, and this was a revolution in computer architecture, was they actually - they dropped the idea of unbelievable rats nests of AND and OR gates, essentially, to try to implement this ever-expanding aggressive instruction set. And they said, wait a minute, computers have sort of flow paths. There's an adder that can get its data from different places, and there's a memory buffer that has the contents of memory, and there's maybe a multiplier that has its input.

So imagine a very different kind of instruction word, probably long, many bits. But the bits in this so-called microcode, they just enable the flow of data along these different data paths at different times. And so just by sort of opening and closing these data paths in sequence, we can implement a complex instruction in multiple small steps so the outside world still sees it as a single instruction. Inside, the microcode has many small steps which it goes through to pull off this complex instruction. So the programmers don't see it on the outside, but the engineers who engineer the microcode, they came up with a whole new way of thinking about how to engineer a computer by...

Leo: I always assumed that all processors had microcode. I didn't realize that was a later invention.

Steve: Yeah, it was. And it was something that was developed through this necessity of

they just couldn't, you know, it was like how are we going to do this with just AND and OR gates?

Leo: Right.

Steve: And so the idea was that you'd have the so-called "microstore" was very fast because it had to run in, like, much faster than main memory. Main memory was chunking along, just reading instructions. But the microcode had to run many times faster in order essentially to have all those little microcycles fit within one major instruction cycle in order to get the job done. And that also meant that the instructions no longer all took the same amount of time. As we said, more complex instructions would take more time. But that allowed this flexibility of instruction length, for example. The microcode, if you were doing a fancy operation like adding two words, both in main memory, and storing them to a third, that was no problem now. That instruction had many more microcode steps in it in order to get the job done.

And so this really changed the way the system was working, the internal design of the computers. And so what that also of course did was create a renaissance in complexity because, when the programmers heard that now there was like a computer in the computer, then they went hog wild with instructions they wanted because it was like, I mean, it was very much like, you know, we all have heard the slogan Apple has, "There's an app for that."

Leo: Right. There's a microcode for that. There's an instruction for that.

Steve: There's an instruction for that. You know, anything these guys, these programmers could think of, they'd say, hey, how about one that does this?

Leo: Right.

Steve: And the engineer said, really? You need that? Oh, yeah, yeah, I needed that yesterday, and I wish I had that. Okay, fine. Well...

Leo: So it does operate faster if you put it in microcode in the chip than if you wrote it. I mean, it's easy to write a linked list in a higher level language, or even assembler. But it's faster if you put it in microcode.

Steve: Well, and, see, that - the perfect question. Because remember that main memory was still very slow.

Leo: Ah.

Steve: And it was very expensive.

Leo: Got it.

Steve: So what had happened was computers turned into interpreters. It was...

Leo: Right. To save memory.

Steve: Well, exactly. So what you had was super powerful instructions that then inside the computer it ran around and pulled it off. But that meant that you were saving main memory and you didn't have to fetch it that often. That is, if fetching from memory was time constrained, and it was, because main memory was slow still, then - it was core. Remember, every time you had to read, reading was a destructive process. So then you had to rewrite what you'd read because reading, the way to read from core is to set everything to zero, and you see where pulses come out in the cores that switched from a one to a zero, meaning that those were ones before you set them to zeroes.

Leo: And now they're nothing.

Steve: Now they were nothing, so you had to reset them to ones again.

Leo: Holy moly.

Steve: So this gave a lot of opportunity for the microcode to run much faster, and it meant that your code density and main memory was very high. Essentially, we'd created an interpreter. And we know that an interpretive system allows very dense interpreted instructions where the instructions themselves are doing a lot.

Leo: Ironically, we do have a constrained hardware environment these days with mobile devices, with cell phones. And they, for the most part, use interpreted virtual machines to save space.

Steve: Well, so, yes. What happened was, as technology moved forward, memory became faster. And more than that, it became cheaper. We just got better production rates, and we got more clever, and we began to not be memory constrained. And at some point some engineers said, you know, let's profile actual code that's in the wild out there that's running. Let's analyze the code and see what instructions are being used. The other thing that had happened during the same period of time is compilers came into use. Back in the beginning it was the programmers writing all this in machine language/assembly language that said, oh, I'd love to have an instruction for doing linked lists. Then I wouldn't have to be writing that all the time.

And so over time compilers came into play. It just made sense to create a higher level language that itself would then create the assembly language, the machine language that would drive the machine. And but here was a problem. Compilers, it turns out, couldn't make use of all this fancy stuff. The compiler, the nature of the compiler - see what I mean?

Leo: Yeah, yeah.

Steve: Yes. Humans could; but the compiler was like, wait a minute.

Leo: I don't want to do that.

Steve: Yeah. Well, it just - it didn't make sense. What had happened was instruction sets had become very specialized. And individual...

Leo: Right. So it's great for assemblers, for people who write, like you, who write machine language. But not for an automated system.

Steve: Not for a mechanized code generator that would sort of always be using the least common denominator. And so...

Leo: Well, that's a good point, too, since not every chip would have that capability.

Steve: Right. So when the computer architects of the past profiled the actual instructions that were being used, they discovered the familiar 90/10 rule. You know, 10 percent of the instructions were used 90 percent of the time.

Leo: Right, right.

Steve: And then they said, wait a minute. If these fancy instructions are not being used, look at the expense in this machine for a linked list instruction. Or these wacky bit search nonsense that we ended up getting talked into by the programmers. Compilers never use these. Yet even though they're not using them, they're in the machine. Every one that we push out the door has very expensive-to-implement instructions that nobody uses because we fired the assembly language programmers, and we hired Fortran programmers or Cobol programmers. And the compilers now do the job that the assembly language guys used to do, and the compilers aren't using any of these instructions, which every single time we push one off the assembly line we're paying for all this microcode store to implement things that no one's using. So there was a complete rethink at that point. And...

Leo: Where does this happen? Does it happen at Intel, or is it happening elsewhere?

Steve: It was happening in - this was sort of - it was happening in universities, actually. The SPARC came from Stanford and Sun, or Sun later. And the MIPS RISC machine was at Berkeley. And the ARM from Acorn was in the U.K. And so that's where this notion - they said, hey, let's reduce the instruction set complexity. And someone said, ooh, RISC, R-I-S-C, Reduced Instruction Set Complexity, or Reduced Instruction Set Count, it's sometimes known as. The idea was that they realized they'd sort of gotten way off track

with these incredibly expensive instruction sets because, they said, wait a minute, let's - hold on. Let's instead kind of go back to where we were. That is, let's have very simple instructions which, now that main memory was fast, had caught up in terms of the speed we needed, and it got cheap.

The other thing is that memory costs dropped through the floor. So you no longer had to have an entire operating system and 20 shared timesharing partitions all fitting somehow into 64K, which actually was done once. No, now we had, at this point, megabytes of memory, easily, so code didn't have to be as dense. And essentially there was this wave of simplification that said, wait a minute, let's go back to - oh, the other thing is that, remember, we went to a - we had variable-length instructions? Well, the only way you could handle a variable-length instruction was in microcode, having microcode that knew that, oh, look, this particular opcode requires these different parameters so we've got to go fetch those now. And that means this instruction has more bytes after the opcode of parameters for the opcode.

Well, all that was thrown out the window. They went back to a radically simplified architecture. One of the most popular is called a load-store architecture, which is now, currently, today, the most popular of architectures, the idea being that you have exactly one instruction for loading something from memory. You have exactly one instruction for storing something into memory. And all the other instructions deal with a much richer set of registers, maybe 32 is the number that most of these architectures have now. So you've got 32 registers. And you can do things like add register one to register two and store it in register three. You cannot add register one to the contents of memory. The other thing, again, in a code analysis, what was seen was that that wasn't being done that often.

So they said, okay, let's try a whole different approach. Instead of instruction, instead of, like, having all instructions able to add and subtract and multiply and do things with main memory, let's have a load instruction and a store instruction. And then everything else, our jumps and our branches and our map and logical things, those only work with the contents of what's in the registers. And we're going to have - oh, that allows us to go back to a fixed length. And that means that every single instruction, like back in the old days, is a single cycle. That is, one cycle. So now that main memory can catch up - or, if not, then caching had come into vogue.

So then there was this notion of they noted that programs tend to sort of stay in the area where they're executing a lot. And so the concept of caching came in. In order to provide a RISC processor that was able to ask for an instruction every single cycle because it was able to execute an instruction in a single cycle, the cache would feed instructions at that rate. And so the idea was lots of little simple instructions executed very fast. And then other things could be done, like you could, if you knew that the instructions were all the same length, you could fetch a block of them ahead of time and bring them into the cache so that they were ready in case the processor wanted them.

So that's the - there's a little bit of, like, coming back where we came from, but with a much more mature understanding of how to get the most bang for the buck. And the major thing that happened is that this rethink of architecture allowed for a dramatic simplification. Now, Intel was, and even to this day is stuck with this insane instruction set, which I happen to...

Leo: How interesting.

Steve: ...know by heart.

Leo: Yeah. The x86. They wanted to get out of it. They wanted to stop.

Steve: Yes. The x86, with its variable byte length between one and 16 bytes, they're stuck. The problem is, the reason they're stuck is backward compatibility. It matters too much, I mean, they've always wanted to, like, break free. But how can they ever do an incompatible chip that, like, didn't run everything? And they've just...

Leo: In fact, they were going to, and AMD was the one that put the screws to them by saying, well, we're going to keep doing x86.

Steve: Exactly, yes. And so what happened was, it was, for example, a company like Acorn, back in the late '80s, that was trying to come up with a successor. Acorn was the preeminent personal computer company in the U.K. There was Sinclair, and Commodore, and then also some U.S. companies had strong sales. But Acorn was the number one. They had a system based on the 6502, which was originally the processor technology - I'm sure that was the company. Although processor technology was a different company also.

Leo: Yeah, there was - oh, my. You know, it's funny how blurred it all starts to get. The 6502 was in the Apple II and the Atari, I remember.

Steve: Oh, yeah, yeah, exactly.

Leo: A crappy chip, by the way, the worst instruction set. Oy.

Steve: Careful, there, Leo.

Leo: I liked the 68000 instruction set, the Motorola instruction set. That was a very clean..

Steve: Well, but that was a whole different generation. It's really not fair...

Leo: It was, you're right. Oh, no no no, I know. But even the Z80 I think was a little bit cleaner than the 6502.

Steve: Well, that, too, was a different generation. What the 6502 was, was incredibly inexpensive.

Leo: Right.

Steve: And it had a beautifully...

Leo: It was MOS Technology.

Steve: M-O-S, that's it, MOS Technology, thank you.

Leo: Thank the chatroom.

Steve: MOS being Metal Oxide Semiconductor, of course. It had a beautiful instruction set. And what that allowed was it had just enough to get the job done, which meant it had a low transistor count, which meant it had a small die size, which meant your yield on wafers of the time, that is, you got many more processors per silicon wafer because the individual dies were so small because the transistor counts were so low. And that meant that made these processors incredibly inexpensive. And that's why Commodore, Apple, and everybody jumped on the 6502. So what the Acorn guys did was they took a similar approach. Because they'd been 6502 users, they took a similar approach to a brand new design, and they were going to do a RISC processor. So naturally they called theirs the Acorn RISC Machine, ARM.

Leo: Oh, my goodness. I had no idea.

Steve: And they ended up with a beautiful, lean design.

Leo: Right, right.

Steve: It was the small transistor count. Now, the other thing that is expensive in large dies and large transistor counts is every transistor you've got not only takes up space, but it takes up a little bit of power. And so the other problem Intel has been fighting valiantly all along is they've got this insane instruction set that they can't get away from which they're trying to make faster and faster somehow. The problem is they are just dogged by power consumption because this complex instruction set requires a huge amount of power, I mean, like, disproportionately so. Just to feed the latent complexity in the instruction set takes its toll. Whereas the ARM, the Acorn RISC Machine chip, because they had the idea, I mean, they came a little bit after the SPARC and the MIPS, so they were able to learn from those designs. But they also came from a 6502 era of understanding low complexity. And interestingly, they were a small company with a small design team. And they didn't have fancy tools. So they also had no choice but to do a simple, small, RISC machine.

Well, even though it couldn't compete then with the SPARC and the MIPS, it competes now because of its low power consumption. And what happened was, in the evolution of Acorn, they ended up renaming themselves Advanced RISC Machines Ltd. Again, ARM. And they're a licensing company that licenses the intellectual property of the ARM RISC core and allows other people - they don't produce them themselves. That's why, for example, Qualcomm has the Snapdragon, which is an ARM-based chip. And it's why Apple with their A4 is an ARM-based chip.

So Advanced RISC Machines is an intellectual property licensing company, licensing this beautiful core which has, because of its - it was always an efficient design, but it was a small design because that's the only thing they could design. But "small" means very inexpensive, just like it did on the 6502; and it means there just aren't that many transistors. The first ARM had only 25,000 transistors, compared to now you've got hundreds of millions of transistors. So you had very low power and, because it was a relatively state-of-the-art design, the right architecture. And that's why everybody who's doing handheld devices has ARM-based technology.

Leo: Makes sense.

Steve: You get a lot of - you get the processing power you want, and it's inexpensive because it's a small die, and it's inexpensive in power consumption because it doesn't have that many transistors hungrily burning up power. And the ones it has are all in use. The other problem with the Intel instruction set is, as we saw, it's inefficient due to the 90/10 rule. So you have all these transistors sitting around not doing much because they just end up being the exception rather than the rule. Whereas this lean RISC design has all your transistors busy all the time, so that's efficient, too. You get a lot of use out of the power that the transistor is using. And that's the story of RISC.

Leo: I love it. And very timely because of course here we are sitting, looking at RISC processors all the time now, in our phones, in our iPad, and all these small devices.

Steve: Yup.

Leo: You know, Android, which is originally running on an ARM processor, is being ported to the Intel Atom processor so that Google TV can run on an Atom-based system. Of course, it runs on a Java Virtual Machine, so it's probably a simple port to make. You just need a virtual machine to do it. But I wonder what kind of efficiency hit they're going to take.

Steve: It'll be interesting to see. In two weeks what I want to talk about, and I think this exhausts me in talking about technology, is the need for speed. Because there's a fascinating story that's the last thing I want to cover on what has been done inside these machines to give us the speed they have. People are not going to believe, I mean, it is - it just grays your hair to think of the engineering that has gone into this. We're going to talk about pipelining and multiple issue and out-of-order execution.

Leo: Ooh, I love that stuff.

Steve: Branch prediction, superscalar stuff. I'm going to lay that out. And at the end of that episode everyone's just going to look at their little, their iPod or their pad or their phone and think...

Leo: Oh, my.

Steve: ...that stuff is in there. And, I mean, it makes you appreciate, I mean, I'm just stunned by what went into the technology that we so easily take for granted today.

Leo: It's really true. It's so complex. And yet it works.

Steve: It does.

Leo: Yeah. Fascinating stuff. I hope everybody's who's interested knows about this and listens. And tell your friends if you think they'd be interested. Steve Gibson is the guy in charge of the Gibson Research Corporation, GRC.com, if you want to find his SpinRite program and all the free stuff he gives away. He's also blogging now: steve.grc.com. And heaven forfend, hell hath frozen over, he's tweeting: @SGgrc, or @SGpad for iPad stories or pad stories in general.

Steve: And for what it's worth, the followers of my tweet stream found out about the Flash problem immediately and...

Leo: That's a good way to find this stuff. It's a good early warning system.

Steve: Yes.

Leo: It really is. I think of it almost as a nascent Internet nervous system where this stuff just goes out.

Steve: And also people who want to listen live learned that I was going to be on Thursday instead of Wednesday the same way.

Leo: That is true, too. Yes, follow him on Twitter, @SGgrc. Steve, next week we will do Q&A. So if people have questions or comments or suggestions, they should go to GRC.com/feedback.

Steve: Please.

Leo: And if you want the 16KB version of the show, Steve makes a version of that available along with transcripts and show notes at his website, GRC.com. So that's another reason to head over to the GRC site. And Steve, we'll catch you next week.

Steve: Talk to you then, Leo.