SECURITY NOW!

Transcript of Episode #235

Machine Language

Description: After starting at the very beginning two weeks ago by looking at how resistors and transistors can be used to assemble logical functions, this week Steve and Leo use those functions to build a working digital computer that understands a simple but entirely useful and workable machine language.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-235.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-235-lq.mp3

Leo Laporte: This is Security Now! with Steve Gibson, Episode 235 for February 11, 2010: Machine Language.

It's time for Security Now!, the show that covers security, privacy, what you need to know before you go online. The man who does this all for us, the great Steve Gibson. He's the guy in charge of the Gibson Research Corporation, GRC.com, the head honcho there. He's also the author of SpinRite, world's best file, rather, hard drive recovery and maintenance utility. And he's got a lot of freebies up there, too, at GRC.com. Good morning, Steve.

Steve Gibson: Hi, Mom. Oh, I mean, hi, Leo.

Leo: Is your mom - does your mom ever watch?

Steve: No. She had enough of me while I was growing up, I think, that's...

Leo: Were you kind of nerdy as a kid?

Steve: Oh, goodness. Yeah, I mean, going on family vacations I had to drag all of my strange gadgets and things with me, just so I had something to play with while everyone else...

Leo: But wait a minute, now, you and I are the same age. When we were young, gadgets were, I mean, we weren't talking Sony Walkmans. What gadgets did you have?

Steve: Well, switches and diodes and resistors and things.

Leo: You probably had - you had a crystal radio, didn't you.

Steve: I was building things. Oh, yeah, I had, you know, I had projects even back then. I have really important stuff to do. I can't really take time off for a vacation, but if you're going to make me go to the beach, fine. I'll figure out something, some trouble to get up to.

Leo: I don't know how your parents felt about that, but I would welcome - I think that's just great. When I see a kid that's got, like, a passion about anything, it doesn't matter, I understand that.

Steve: Mom was worried.

Leo: Was she?

Steve: She was like, I'm not quite - we're not quite sure what he is, so...

Leo: Well, you were a little - just ahead of your time, that's all.

Steve: Yeah, I was definitely trouble, so.

Leo: So today we continue our series on building a computer from scratch; right?

Steve: We're going to do something so cool. The goal, which we will achieve, we've achieved it before, is to so thoroughly and completely demystify and explain something that our listeners are going to end up thinking, wait a minute, that's all there is? That's it? That's what the big deal is with programming in machine language? That's simple. And it's like, yes, it actually is very simple.

And so that's what we're going to do today, is I'm going to - we're going to essentially design a machine, now that we understand from two weeks ago how gates are built up from resistors and transistors. We're going to look at what machine language actually is and why it's no big deal. It's really not. It's how you use it and build on it that gets complicated. But that's true of any computer language, anytime you're taking little, tiny, itty-bitty steps. And it takes a lot of them to make something go. Well, the steps themselves are simple. It's the building on top of them that gets complicated. But the steps are easy, and we're going to understand that by the end of this podcast.

Leo: I started programming when I was - I wasn't a kid. I was, I think, personal computers didn't come out till I was in my 20s. So I was maybe 25. And I started in Basic, as a lot of people did, but very quickly got to, on the Mac, got to assembly language, 68000 assembly. And 68000 assembly is beautiful. It's very clean, not like the 8086 IBM assembler language.

Steve: That's true.

Leo: Which I played with a little bit. But I think it was a great discipline to learn intimately how a computer works. If nothing else. I mean, I know you still program in assembler.

Steve: I do by choice. My feeling is, somebody who really understands, no matter what language you're programming in, if you understand something about what's ultimately happening in the basement, you're just better able to make choices. Programming, coding, is all about making choices. You're trying to solve a problem. You need to cast the solution in the way you express things to the computer. And there's a huge amount of freedom, which is why programmers, I think, program, is they really like the freedom and the responsibility, that there's - it's just so open-ended. But...

Leo: It's a fully creative act. It's just like a painting or anything else. Although, well, it's math and science, I mean, math and art; isn't it.

Steve: Well, and with it comes responsibility. I mean, we've heard a lot this last couple weeks about what happens when you don't quite get your braking software correct on a Prius.

Leo: Oh, yeah. That one-second delay could kill somebody.

Steve: Yeah. There's a bug in the Prius software switching from the physical braking to the regenerative braking. And that wasn't quite synchronized properly. And so, you know, I'm shivering a little bit as I hear this because we know, anyone listening to this podcast for, well, the last four and a half years will have acquired an appreciation for how difficult it is to get software exactly right. And if you've got software controlling the braking of your car, which is obviously what we now have, and arguably cars are going to become more software controlled moving forward rather than less, then it becomes really important to get it right.

Leo: Yeah, we're kind of like airline pilots now with fly-by-wire technology. It's not, you know, you push your foot on the brake, it doesn't actually anymore move the calipers.

Steve: Yes, there's nothing connected.

Leo: Right.

Steve: Exactly [nervous laughter].

Leo: Right [nervous laughter]. Well, that's coming up. We've got security news, and there's some big, big news, too. All right, Steve. Shall we start with the patches?

Steve: Yes, well, we had a quiet January for Microsoft, for which they have fully compensated for February. We're just after the second Tuesday of the month in February. Microsoft released 13 bulletins covering 26 vulnerabilities. 11 of those bulletins affected Windows, and two were for Office products. There's even one for Microsoft Paint. If you open a malformed JPG in Microsoft Paint you can have your computer taken over. It's like, okay, well, maybe that's not a big problem. But it's good to have that patched anyway.

Leo: Yes.

Steve: Most of them were rated highly exploitable in Microsoft's kind of wacky exploitability index, which is their way of saying how likely it is that the vulnerability could actually be turned into something that Windows users need to worry about. Interestingly, one flaw was in the new IPv6 stack in Vista and Server 2008. It's possible to ping an IPv6-enabled Vista or 2008 Server machine and take it over. So that's being fixed. So basically, rather than going through every one of these, there's sort of no point. I mean, most of them are remote-code executions. They're scattered throughout the operating system.

Significantly, the problem we talked about last week which was shown at the Black Hat Conference in D.C. the week before has not been fixed. It is still hanging out there. Presumably it just came up too soon for Microsoft to get this thing fixed. And that was the exploit that turns your Windows machine into a public file server, which is not something you ever want.

Leo: Yeah, that's what got you first started on this whole security thing, is that open file shares.

Steve: Well, exactly, it's when Microsoft's policies allowed that to happen too easily. So, you know, now with the firewall running they've closed that down. It turns out that there's a way to trick IE into opening a connection to a remote bad location that gives it access to your entire hard drive. This was demonstrated at the Black Hat Conference. The details are not public. The guy who figured it out is waiting until Microsoft fixes it, and then he'll show everyone how he did it. So I'm sure Microsoft is on the ball with this one. I'm sure he's got their attention, and we're not going to be waiting six months because at some point this guy's going to say, okay, I'm done waiting, here's how you do this. Sorry, Microsoft. So anyway, everyone wants to run through Windows Update, get themselves current, because there was a bunch of things that need to get fixed.

Also I wanted to mention that, just in terms of calendaring, Microsoft's support and security terminations are approaching. Microsoft recently reminded the world through some of their various newsletters that, as of April 13, so a couple months from now, Microsoft will no longer be issuing security updates for the original version of Vista. I don't know that that's really that important because certainly anybody using Vista would have moved to SP1 or SP2, hopefully. So Vista will end up, the security updates will end up being terminated. And then we have six months before Windows 2000 completely gets - all support for Windows 2000 stops. And even XP SP2 will stop at that time. So certainly SP3, which is the current one, should be running by that time. And then you'll be able to continue getting security updates for that.

Also in security news, Apple's iPhone and iPod Touch had multiple vulnerabilities fixed recently. Anyone using their iPhone should make sure they're at 3.1.3, which is the current release, both for the iPhone and the iPod Touch, from Apple's site. There were a bunch of things, none of them very good news. There was a problem in their Core Audio system. Apple says the impact is playing a maliciously crafted MP4 audio file may lead to unexpected application termination or arbitrary code execution, meaning that you could basically hide a trojan in an MP4 audio file.

Leo: That's really bad. I mean, we always say you have to execute a program to get onto somebody's system. But what can also happen is malformed data - and we've seen this on Windows before, like a JPG or a PDF malformed - can allow a hacker in. But, boy, you don't want to see that on a phone.

Steve: Exactly.

Leo: Especially with a built-in player; you know?

Steve: Exactly. And in fact they had a similar problem in their ImageIO library where by Apple's own statement they said viewing a maliciously crafted TIFF image may lead to an unexpected application termination or arbitrary code execution. And they had a couple problems in WebKit and a problem in recovery mode. So those things are fixed. But exactly as you say, Leo, these are important, especially in something as connected as the iPhone. So all users need...

Leo: I think increasingly, as you see smart phones become so prevalent, you're going to really see more and more attacks on this because it always connected. There are vectors that you have through Bluetooth and stuff as people carry it around, or through hotspots. And really, isn't that where you want to be is on the phone? Almost better than a computer.

Steve: Well, and we've seen instances where there were, like, malicious text messages, where you could just text somebody. And the phone, in the process of the phone's receiving this specially crafted text message, which there was no way for it to block, there was no way to turn that off, it would take the phone over. So, yeah, these things are definitely important. And staying current with known patches is very important because what the bad guys of course are doing now is when the patches come out, they analyze the fix to reverse engineer the problem. And then they exploit the problem on the assumption, which is unfortunately too often true, that not everybody has yet installed the fix. Thus...

Leo: The good news on the phone is, as soon as you plug it in, it says there's an update, and it's pretty trivial to do it. Well, good and bad because then there is a whole group of people, not huge, but a whole group of people who have jailbroken their phones. And of course they don't patch because the patch immediately breaks the jailbreak.

Steve: Right.

Leo: And so they wait until the jailbreakers say, okay, it's safe to update now, we have a fix for it. And that means it could be a week - could be a day, could be a week, could be months before you can patch. And so that really does open a vulnerability.

Steve: Yeah.

Leo: I stopped jailbreaking because I - for many reasons, but that was one of them. And now especially.

Steve: CNET reported that the FBI has been pressing ISPs again - this kind of keeps coming up through the years - again pressing ISPs to retain records of every URL visited by their customers. Drew Arena, who's Verizon's vice president and associate general counsel for law enforcement compliance, said, quote, "We're not set up to keep URL information anywhere in the network. If you were to do deep packet inspection to see all the URLs, you would arguably violate the Wiretap Act." So there's some pushback on this. But we're really seeing law enforcement working to try to get ISPs to log everything that their customers do.

Leo: This keeps coming up. I thought this was - remember Carnivore, the FBI thing that they wanted ISPs, in fact, I think many ISPs actually are running it?

Steve: Yeah, and in fact a guy named John Seiver, an attorney at Davis Wright Tremaine, who represents cable providers, said that one of his service provider clients had experience with a law enforcement request that required the logging of outbound URLs. He said, "18 million hits an hour would have to be logged," which is a staggering amount of data to sort through. The purpose of, in this case, of the FBI's request was to identify their customers who visited two specific URLs to, quote, "try to find out who's going there." So that's really scary from a pure technology standpoint, the idea being that the FBI would say to an ISP, give us a list of all the people, your customers, who visited a specific domain or a specific set of URLs.

The problem with that, that I have with that from a technology standpoint - and as you know, Leo, we don't control, users, end-users do not absolutely control the URLs that our browsers go to. We click on links which, you know, largely we're not clear about necessarily what the link is. If you look down in the URL monitor field of your browser and hover over the link, you can generally see. But JavaScript can also obscure those so that you don't get a URL report for where you're going to go. And even when you do, you bring up a page that inherently contains accesses to other resources which could easily be bad, coming from domains that you don't visit, but your browser does.

So I don't know, this whole thing is just very troubling. It's, I mean, I totally understand from the law enforcement side the frustration they have. But it seems to me saying to a service provider, we want the identities of everyone who's ever gone to such-and-such a domain, that really seems like overreach.

Leo: Yeah. This is that - it comes back to that whole Patriot Act thing where people are willing to trade liberty because they are afraid.

Steve: Yeah.

Leo: And so it's easy, especially now, and with a war on terror going on and stuff, to kind of push this stuff through. But, boy, it just, it does, it scares me. I hate to see that happen.

Steve: Now, I wanted - I picked up on a little bit of news that I knew that our listeners would jump on and start sending me email because this sounds really bad. And if nothing else it's really interesting. And that is the news that the Trusted Platform Module, TPM, that we've talked about many times...

Leo: And trusted.

Steve: And trusted, which is installed on the motherboards of pretty much all current laptops and now many desktop machines...

Leo: Certainly all business laptops. It's on my Dell business laptops.

Steve: Yup.

Leo: Don't tell me. It's cracked?

Steve: Well, of course.

Leo: I thought this was uncrackable, this couldn't be...

Steve: Sort of. Okay. Chris, a very talented and skilled hacker, a hardware hacker by the name of Chris Tarnovsky, presented sort of a crack at that same Black Hat Conference that we were just talking about. Let me tell you, let me explain what he did. It took him six months of work, during which he did nothing else, and a tremendous amount of skill. Using off-the-shelf chemicals, he soaked TPM chips in acid to dissolve their hard outer shell.

Leo: Oh, this is how you have to look and see what's inside. You have to take the package off.

Steve: Literally. He removed the package. Then apparently there's actually a mesh wiring which is around the TPM inner core, specifically to provide RF shielding, which I think it's very cool. So he used rust remover in order to remove the mesh wiring to expose the chips' inner cores and then used microprobes on the circuitry, on the actual physical circuitry face, to monitor the signals passing inside the chip. And of course, once he found the right spots, he found the signals just upstream of the cipher and so was able to get the plaintext out of the chip that way.

Well, okay. Many people were distressed that this was possible. I'm glad that it was done because it says to the guys making the TPM chips - n this case it was Infineon, which is like the number one supplier of these chips - it says that, I guess, they need to go a little further toward physically protecting the chip. On the other hand, I mean, this is a far-out, extreme instance of hardware hacking. What I want is for the chip in my laptop to keep some keys confidential so that when I swipe my finger, that's authenticated in the chip, and then the hard drive is unlocked, and I'm able to go about my business. I guess what this means is that, if someone really, really, really, really wanted to get access to my hard drive, and they had forever with my laptop, that is, long enough to surgically dissect the chip and pull those secrets out of it, that it is possible. I don't think...

Leo: It's not common, not likely.

Steve: Yeah, I would have never stated that it was not possible.

Leo: Right.

Steve: The good news is...

Leo: So is this different for every chip? I mean, does he have to do it to my chip, to crack my chip?

Steve: Exactly.

Leo: Oh, okay.

Steve: Yeah.

Leo: So that's fine. I mean, I don't expect somebody to take my laptop, apply acid and Rust-Oleum to it and then say, oh, I've got you now. By that time...

Steve: Yeah. So, and even then he said that - he said he still had to crack the internal algorithms, which was a huge problem, and there are traps programmed into the chips' software providing another layer of defense.

Leo: So this is actually pretty good news in the long run.

Steve: It's really good news. Now, and as I said, I'm a little glad that this came out because presumably the chip manufacturers will go, oh, guess we've got to go a little further. I mean, basically what they want to do is some sort of self-destruct technology, I mean, literally, where the act of exposing this to probing renders it unusable. And it sounds like that's the one thing that isn't quite there yet is, I mean, literally like Mission Impossible-style self-destruct, where it just will not function any longer if this has been done to it. So I imagine there's a way they'll be able to do that, now they know they have to.

Leo: Right.

Steve: Thanks to Chris's work.

Leo: Right. Very interesting, though.

Steve: In troubling news, two trojans were found in Mozilla-hosted plug-ins for Firefox.

Leo: Oh. This was only a matter of time.

Steve: I know. I mean, Mozilla, to their credit, goes through extensive efforts to scan the plug-ins to make sure there's nothing evil in them. They received a report that something called Master Filer had a trojan which had to have come from them. They checked again, and none of their 20 different scanning solutions found it. So they added two more, which did find Master Filer. Then, when they rescanned their entire library of Firefox plug-ins, they discovered one other, which was a Web Video Downloader from a pretty neat company called Sothink that is a Chinese company that was sort of the leader in Shockwave Flash decompilers and now actually has a bunch of authoring tools and other good things. They're, you know, it's not at all clear how these trojans got in there.

But anyway, they were found. And Mozilla said that 4,000 copies of the Sothink plug-in were downloaded. And the dates are strange because they say February 2008 to May 2008. So that was, of course...

Leo: That's a while ago, yeah.

Steve: And the Master Filer was installed about 600 times from September of '09 to January of 2010. So I'm not really sure what those numbers say, but that's what was in the report from Mozilla. So it's certainly good that they added additional testing. And not a huge user base was installed, but it's been fixed. So if anyone does have - and Mozilla has said, if you have the Master Filer plug-in and the Web Video Downloader v4.0 from

Sothink, you certainly ought to uninstall it and then run any AV software you've got. And then update, if you want to put it back in. The current versions on the Mozilla site are now fixed.

Leo: So was this Sothink's fault?

Steve: I don't know. We've seen instances where something has gotten into the production systems of software makers such that they shipped software unbeknownst to them that contained something malicious. I don't know enough about the way plug-ins are posted on Mozilla's site to know whether, you know, what the channel was and whether it came infected from Sothink. They claim, you can imagine, vehemently that that was not the case, that there's nothing wrong with their v4.0 Downloader. But who knows. And it does sound like it was a couple years ago in any event.

Leo: Yeah. But I imagine there are people listening who have it, so.

Steve: Yeah, yeah. And one last little bit of randomness. I saw a report that was sort of interesting. Trusteer's Rapport browser security service analyzed four million users of this security service and determined that just shy of half, 47 percent of all these four million users are using the same username and password to log onto multiple sites, including their banking logons. And that of course represents a substantial danger. It's one thing to use the same logon both for Twitter and Facebook. But you don't want to share those logons with BofA and Chase and so forth, your banking credentials.

So I just sort of, having seen this, I wanted to suggest to any listeners who have said, yeah, yeah, yeah, we know we're not supposed to use the same username and password, but it's too much of a pain to have separate usernames and passwords for everything, so we're going to take our chances. I just wanted to suggest that the one exception, if it hadn't occurred to you before, would be to consider that not all of your logons are equally important, and that those which are extra-sensitive really do demand their own username and password because it is the danger of a Facebook or Twitter account being cracked. And we hear about that happening in the news all the time.

The idea would be that the bad guys could assume, hey, if they know that half of users are sharing usernames and passwords across accounts, that they can look at a system's cookies in order to see what your banking credentials are because your bank's got cookies it's left behind on your computer. That tells them where to try to hook up and also provides credentials because cookies are used for easy logon. So that really does create a path for malware or bad guys to logon as you and do bad things to your bank accounts. So if you haven't changed your most important logons to something different from the most popular logons, I really think it's worth doing.

Leo: Very good, Mr. Gibson.

Steve: Um...

Leo: Yes?

Steve: One of the things that was pending was an analysis of LockNote.

Leo: Right, right.

Steve: Which is a very cool little app that I talked about months ago. And I promised last week that I would analyze it, since it was open source, with the source published over on SourceForge; and, more importantly, since they provided no documentation themselves about what it is they were doing. They just said "Trust us," and it's like, okay.

The good news is, I did analyze the source. And these guys deserve the gold medal/blue ribbon for doing the right thing security-wise. The passphrase which the user supplies is concatenated with its length. And that is hashed through an SHA-256 hash to produce a 256-bit key. So that's exactly what you want. That key is used to drive an AES-256 cipher, so that's the Rijndael cipher with its maximum 256-bit length key. So it doesn't get any better than that.

Then they use a very strong cryptographic algorithm, the so-called cipher feedback algorithm or cipher feedback mode, CFB. What that does is, it takes an initialization vector and encrypts it under the key, which as we just said is derived from hashing the user's passphrase. Then the output is XORed with the plaintext which is being encrypted to create the ciphertext. That ciphertext is then fed into the next round of encryption under the same key. The output of that encryption is XORed with the next block of plaintext to create the next block of ciphertext and so on.

What this does is it creates a chain of dependence from the beginning all the way through the end, so that any change in the ciphertext ripples through to the end. And they use a very good, cryptographically strong, pseudorandom number generator for the initialization vector such that, even if you were to encrypt the same text again, that is, the way this LockNote works, it builds an EXE which someone who you send it to, or yourself, you simply run it. And if it sees that its contents have been encrypted, it prompts you for a passphrase. So you put the passphrase in. It hashes it and then runs through the decryption process. They also use a full cryptographic-strength MAC, Message Authentication Code, to verify that it hasn't been tampered with and that no changes have been made.

So they did everything. It is bulletproof. These guys clearly know their crypto, which is great news. And having looked at this, I can tell our listeners that this is as secure as it gets. I mean, everything was done right.

Leo: That's great. So use it.

Steve: Use it without fear.

Leo: Without fear.

Steve: I did have sort of a fun SpinRite story to share with our listeners from actually someone who's not far away from us in Claremont, California named John Irvine is his name. And he sent the note with the subject "SpinRite Saves My Free Hard Drive." He said, "Steve, I just wanted to let you know about my SpinRite experience. This past week, December 18th I believe, my laptop had slowed down. A little history: I got the hard drive from a friend who took his Toshiba laptop to Best Buy for warranty service." Well, we know, we've heard recently what they do about that.

He says, "They took out the old drive and put in a new drive and ran System Restore. He called me from Best Buy and asked if I could get his data off. I told him to bring it to me, and I'd try. I set the drive up on a desktop with an adapter, and the drive came right up. So I retrieved his data, and he let me keep the drive. This was an 80GB drive, and I currently had a 40GB in my laptop. So I swapped the drives, ran my System Restore on my Dell, and it worked perfectly.

"Fast-forward 18 months, and my laptop was running very slowly. So I stuck in my System Restore disk as my laptop has duplicate data from my desktop. Well, the System Restore disk stopped the formatting at 91 percent. I tried once more, and 91 percent again it stopped. So I got out my Ultimate Boot for Windows CD and started that up. It formatted to 91 percent and stopped.

"Well, it was now time to buy SpinRite. I had been meaning to, but did not have a need until now. So I purchased the software, burned to a CD, and stuck it in my laptop. It went to work, and in about three hours it had gone through the first 90 percent of the drive. So I was very interested to see what it would do at that critical 91 percent. Well, as it arrived at 91 it started working, and working hard. It stayed on 91 percent for 30 hours, then moved to 92 percent, 93, 94, in about 25 hours. Then it got stuck at 95 for another 30 hours."

Leo: Oh, man.

Steve: "Then, finally, it hit 99.4 percent and stuck at that point for 24 hours."

Leo: I admire his faith because, to be honest, I would have rebooted a long time before that. That's amazing.

Steve: He says, "At hour 110:06:09 it finished its scan, and my hard drive now works perfectly."

Leo: Wow.

Steve: "I do not know if this is any kind of a record, but I was impressed that it stuck with it through the whole process. Thanks for a great product. John."

Leo: I'm, frankly, impressed that he stuck with the whole process. SpinRite has no choice. He's the guy that's...

Steve: SpinRite will go until you say quit or until it succeeds. And it succeeded.

Leo: It's pretty amazing, really.

Steve: Yeah, it's neat.

Leo: But that - and it isn't a record because didn't we have an email from somebody a couple of years ago that was like, it took six months or something?

Steve: We've got people who are just sort of so fascinated by the process, they'll, like, set it up on some computer they're not using at all, just to sort of see if it can, you know, turn lead into gold. So...

Leo: And it often does. So just to clarify for people who haven't listened to the show as thoroughly and as assiduously as I have, what's happening is SpinRite will continue to read a sector until it gets the data off of it, which sometimes is one time in a thousand.

Steve: It can read it pieces at a time. It can read sectors that could - it'll even read what the drive won't read by pulling the data off and then essentially reverse engineering what the problem must be in order for the ECC to get corrected. So it does all kinds of things.

Leo: And the operating system will not do that. I mean, it just gives up after a few tries.

Steve: No. The operating system sort of says - it basically stubs its toe and says, okay, you can't have that file back.

Leo: And for understandable reasons. You wouldn't be thrilled if your operating system waited 110 hours to come back to you and say, okay, I loaded the file. That wouldn't be good.

Steve: Yeah.

Leo: No, we don't want that.

Steve: No.

Leo: This actually ties very well into our commercial. Then we're going to get on with machine language. Are you going to derive it from first principles?

Steve: Yeah.

Leo: Sure.

Steve: We're just going to - yeah, why not?

Leo: Yeah, of course, says Steve Gibson.

Steve: We have an hour. We have an hour.

Leo: I can do that.

Steve: Yeah.

Leo: Yeah, we're going to just relive the entire history of computer technology over the last 40 years in - we'll do it in an hour, no big deal. Let's get to machine language.

Steve: Okay.

Leo: Where do we start?

Steve: Well, two weeks ago we sort of laid down the foundation by demystifying AND and OR and NAND gates and how you could cross-connect two inverters that would create a little memory cell which could remember if it was set to one or set to zero, a so-called "flip-flop." And I wanted to convey to people the sense from back in 1960, essentially, for the number of components that were required to do even the simplest things. So now we have gates and the ability to create a register of individual bits which can be read and written.

So how do we, literally, how do we make a computer? And I know from talking to people so much, there's sort of this mystique about assembly language and machine language, as if, like, you have to be some galactic guru in order to understand that. It's like, oh, that's like really deep voodoo. And I'm going to use C or Perl or PHP or Python or something. The truth is that what's actually happening down at the hardware level is really simple. And, I mean, I'm going to demonstrate that now by looking at and sort of designing, developing right now with our listeners a completely workable, usable computer, using only what we understand, no magic. And I believe that, once we've gone through this exercise, sort of wiping the slate clean and just saying, okay, let me just think about this, people are going to end up thinking, well, okay, that's it? And the answer is yes. I mean, it's not that big a deal.

So we have memory for any machine. Back in the early '60s we had gone from drum memory to core memory. Drum memory was sort of the predecessor to core, the idea being that you'd have a magnetized drum that was spinning and literally use the impulses coming off of the drum as the contents of the computer's memory. Thank goodness that was replaced when this concept of cores, little tiny doughnuts, essentially, that are magnetizable in either a clockwise or counterclockwise direction. We've talked about it once before, the idea being that this memory could store a one or a zero based on whether the individual little doughnut was magnetized in one direction or the other. And you could tell which direction it was magnetized in by forcing them, like a set of them, all to zero. If they were already at zero, nothing would happen. If they had been at one, then the act of switching their direction would induce a pulse in a so-called "sense wire," and that would tell you that, ah, we just moved that one from one to zero.

Well, that was called a "destructive read" because the act of reading the contents destroyed the contents. We wrote zeros to everything, getting pulses out of those ones that switched from one to zero. Which meant that, unless we wanted to leave the zero there, we needed to rewrite the original contents in order to put it back, which is what these memories typically did.

So let's imagine that we have memory, core memory, which is nonvolatile, meaning that it just - we can magnetize these little cores, and they'll stay set that way. And we have a way of reading out the contents of a location and getting the ones and zero bits that are there. So the first thing we need to have is what's called the PC, the Program Counter, which is a - it's a counter which increments one at a time, reading out the contents of successive words of this memory. Now, the word length can be pretty much whatever we want. There were word lengths back in the beginning of as much as, like, 36 bits, sometimes even more. The early DEC machines were 18-bit word length. And people are used to thinking these days in terms of 16 bits or 8-bit bytes. We know, for example, that the Pentium machines were 32-bit machines, and of course we now have 64-bit systems. So these are the - currently there's been complexity added to what so-called "word length" means, which we're going to actually talk about in two weeks. In two weeks we're going to talk about all of the stuff that's happened since. But back in the beginning the word length was whatever the designers wanted.

Now, there was pressure on keeping it short because everything cost so much. Remember that, back then, this was before integrated circuits. So a bit that you had was a bunch of circuitry that you had to pay for every time you made one of these machines. So, sure, the programmers would like more bits because that allowed them to store more stuff. But the management was saying, wait a minute, we can't afford this many bits. So there was sort of a compromise. So if we look, for example, we don't really have to worry about specifically, but just sort of imagine you had 18 bits because that's where the first machines of this era sort of landed, 18 sort of being a compromise of different pressures, cost and capability.

So we have this program counter which will address the memory sequentially, basically stepping through it. So say we start at location zero. So out comes 18 bits into a register which we call the "instruction register." And it's just, it's one of these registers made out of individual bit memories, which we talked about last week. And they're all expensive, but we can afford 18 of them. So this instruction register holds the data that we just read out of a given location in memory. So what do we do with that?

Well, there's essentially a subdivision of the bits into different purposes. And a term that probably everybody has heard is opcode, the operation code. And sort of traditionally, the opcode has been on the left of one of these long words. So, for example, in the case of this computer we're making, we'll dedicate some number of bits to the opcode. So, okay, what does that mean? What things do we want to be able to do? We want to be able to load and store memory. We want to be able to add and subtract and maybe perform some logical operations.

Now, we're performing these against something called the "accumulator," which is another register. We had the instructor register; now we have an accumulator, which is sort of our scratch pad, so that that's the main working register where the data moves through where we perform these operations. So, for example, if an instruction said load a certain location into the accumulator, then the computer would transfer the data in a given location in its memory into the accumulator. And if another instruction said store that somewhere else, the computer would store whatever happened to be in the accumulator now into the location specified.

So we need to be able to perform some operations on the data in this accumulator. And sort of - so this is everything is centered around the accumulator, with the rest of the hardware sort of all existing to serve the purposes and needs of this accumulator. So if we had an opcode of, say, 5 bits, well, we know how binary works. We know that each bit gives us twice as many as we had before; 5 bits means that there's 32 different combinations of 5 bits. So if we think of those as sort of as the verb of this instruction, we could have 32 different things. And in fact the PDP-1 was an 18-bit computer that did have a 5-bit opcode. But back then 32 verbs, 32 actions that you could specify turned out to be more than they ended up being able to use.

So as the DEC minicomputers evolved, in fact with the very next one, which was the PDP-4 - there was no 2 or 3; the 4 and the 7 and the 9 and finally the 15 were the 18-bit lineage - they dropped the opcode to 4 bits, which is where they stayed for quite a while, for many years. So 4 bits gives us 16 different verbs, 16 different things we could do. So, for example, the opcode, meaning the first four bits of this word, might be 0000 or 0001, 0010, and so forth. Each combination of those 4 bits would specify a different action. And just one simple action. So absolutely one of them would be load the accumulator with something in memory. Now, where in memory? Well, that's where the rest of the bits come in.

Leo: All right, Steve. So we're building a machine language. And it really is based on kind of the architecture of the CPU; isn't it?

Steve: Well, I think what's significant, the point that's worth making is that even though I'm talking about an architecture that is 50 years old, this is still today exactly the way computers work. What I'm talking about is a simple CPU, a simple Central Processing Unit. But the fundamentals haven't changed at all.

Leo: Probably not even since Alan Turing imagined how a computer would work in the '40s.

Steve: Right.

Leo: This is the fundamental way a computer works.

Steve: So we've got a 16-bit word. And the left-hand 4 bits are allocated to the opcode, which leaves us 14 bits for the address. Meaning that the word is two parts. There's "what to do," and then the second part is "and what to do it with." So a 14-bit address gives us 16K words. If we think of, like, 10 bits is 1K, 11 is 2K, 12 bits is 4K, 13 bits is 8K, 14 bits is 16K. So the right-hand 14 bits provides the address, sort of the address argument for the opcode verb.

So say that the opcode 0000 stood for "load the accumulator." So when we fetch this 18-bits instruction into the instruction register, there's some logic which looks at the combination of bits in the opcode and essentially does this one simple thing that the opcode specifies, like load accumulator, if all four of those bits are zero. And so what that means is that that 14-bit argument is used as the address to fetch another piece of data from memory, different from the instruction. We fetch the instruction from where the program counter is pointing. Then we fetch the data from where the 14-bit argument of that instruction is pointing and load that into the accumulator.

So the opcode 0001 might be "store accumulator." And then the 14 bits following it would specify where to store the accumulator. So with those two instructions we have the ability of picking up data from somewhere and storing it somewhere else, moving the data from one place to another in memory. We might - we would certainly have an instruction called ADD. That might be 0011. And what that would do is - and then the 14 bits that follow would specify where to go to get the data to add to what's in memory. Again, it would - and this class of instructions are collectively called "memory reference instructions" because each of those opcodes references memory. It loads it; it stores it; it adds it to the accumulator; it might subtract it from the accumulator; it might AND it against the accumulator or OR it with the accumulator. Basically very simple, simple bit manipulations against the accumulator.

Now, the computer is useless to us unless it's able to have some sort of I/O, some sort of input/output. So one of those instructions, which would not be a memory reference instruction, would be an I/O instruction. Maybe that's, like, 1111, all the way at the other end, the 16th instruction, 1111. That would - it would be formatted differently. That is, the memory reference instructions were all an opcode followed by 14 bits that specified where in memory to do its thing. Whereas the last instruction, 1111, that's an I/O instruction.

So the rest of the 14 bits might, for example, specify an I/O device. Many of the early computers had, like, you could attach up to 64 devices. Well, 64 is another power of 2 which you require 6 bits to specify. So there might be a field in those remaining 14 bits that is a 6-bit I/O device number, meaning the teletype, the mag tape, the card reader, the card punch, whatever device it was. And then some of the other bits might be start the device, stop the device, read the device, write the device, different bits that are about input/output rather than, well, because those apply to that specific instruction. So what we see is we see that there's always a field in the instruction word for specifying the operation. And then depending upon that operation, the remaining bits provide arguments of one form or another to it.

Now, at this point we've got a computer which is able to move through memory, incrementing its program counter once for every instruction, and reading what's there and causing something to happen. Read, load and store, input something, output something. The problem is, it just goes in a straight line. And while that's certainly what you want some of the time, one of the things that computers do is make decisions. And that requires altering the normal linear incrementation to jump somewhere else.

The way this was done then, and even now, was to have a skip instruction, the ability to skip over a word in memory. Even though that wasn't very powerful, it was powerful enough because what you might have had, and certainly would have, one of our instructions. We talked about load and store and add and so forth, well, one of those, like instruction eight - 1000 - that instruction could be the jump instruction. And so when we load the instruction in the instruction register, and the opcode is 1000, that is, the first, the left-hand 4 bits is that pattern, well, the argument to that instruction, the other 14 bits, is the address we want to jump to.

So all the computer does is it loads that 14 bits into the program counter. So that instead of the program counter incrementing one at a time, we've just replaced the contents of the program counter with the 14 bits in the jump instruction. Which means that the next instruction we fetch is at that location. We've just jumped our program execution to a different place. That's all there is to it.

And so the way the skip comes into play is that, if we tested something, like say that one of our instructions was skip if the accumulator is zero, or skip if the accumulator is not zero, that kind of thing, well, if we were to subtract two items, and they were the same, that is, if they were equal, then the result would be zero. So that allows us to determine if two things are equal or not. And if we had an instruction that said skip if the accumulator is zero, then the instruction it's skipping over would be a jump instruction, which is - this is all a very simple way of implementing the control of the program's flow, so that if the two things we were comparing were not the same, the accumulator would not be zero, so we would not skip the instruction that follows. That instruction that follows would be jump completely somewhere else, so that if we don't skip, then we land on that jump instruction and go completely somewhere else. If the accumulator was zero, we skip over that jump instruction.

And all skipping means is, instead of adding one to the program counter, we add two, or we add one twice, which is actually how these machines worked back then. And that just causes us to skip over a jump. So essentially that means we can branch to anywhere we want to in memory or continue on our way, which gives us, even though that's very simple, that gives us enough power to allow machines to make decisions. And we've got input/output; we've got math; we've got the ability to transfer data from one location in memory to another. Those are all the essentials of the way a machine functions. That is machine language.

Now, the one layer of humanity that's put on top of that is what's called "assembly language," which is nothing but naming things. For example, you create sort of a so-called mnemonic for the different instructions. So, for example, load the accumulator would be LDA. Store the accumulator, STA. You want them to be short because you're going to be typing them a lot. Remember that you end up using lots of little instructions in order to get something done. And then the only other thing really that assembly language does, it allows you to name locations in memory.

So, for example, you might say LDA, for load accumulator, current score. And current score would simply refer to a, like a variable essentially, a location in memory that you had labeled "current score." And then if you did STA, store accumulator, new score, well, it would first load the current score into the accumulator, and then store that into a different location called new score. So really that's all we're talking about is some simple abbreviations for helping sort of remember and use these individual instructions and convenient labels for locations in memory so that you're not having to remember, oh, that's in location 329627. I mean, who can do that? So instead you just, you label that location with an English, an alphanumeric phrase of some sort, and then you refer to that location by the phrase rather than by its actual number.

And in fact you don't care what the number is. That's one of the things that the assembler will do for you is you just say I need memory called these things. And it worries about where they go because it doesn't really matter to you as long as they're consistently referred to. And that's the whole process. That's machine language and assembly language. And that's the way it was 50 years ago, and more or less that's the way it is now.

Leo: Very cool. It's amazing, really.

Steve: It is. We referred to it the other day as a dumb box of rocks that was just very fast.

Leo: Exactly. And this is - I think that was the most valuable thing about me learning how assembler works is you see every individual thing it does. And so you see exactly that. That's the lesson, is it's not doing very much. It's doing it fast.

Steve: It's why I like it, because nothing is hidden.

Leo: Right.

Steve: That is, there's nothing going on underneath that. One of the problems that I see programmers having is they assume that the compiler, like a C programmer is expressing much more abstract things. For example, when you're dealing at the machine level, you are truly dealing with fixed numbers of bits that you're moving around under your command. When you abstract that a lot, you're now talking about sort of like double-precision something. But the details matter. And it's where the programmer assumes that something is going to be done for him or her by the compiler that the compiler doesn't agree with. The compiler says, no, that's not what you told me to do. I'm going to go off and do this. So that kind of miscommunication in assumptions is where a lot of problems crop up. And for me, by dealing with it, by insisting on actually doing the individual small little bite-size pieces, there's no room for argument.

Leo: Yeah.

Steve: I mean, when I make a mistake, it's mine. It's because I told the computer, move this chunk of bits over here, and that was the wrong place to go. It's not that I told it something, and it did something different.

Leo: Yeah. Well, doesn't mean there are no bugs or surprises. I mean, because humans may think they're saying one thing and the computer think another. But it's much less ambiguous.

Steve: Yeah.

Leo: I mean, it's pretty clear. And I would guess there's kind of fewer interactions. Although, I don't know about you, but as I used assembler, I built larger and larger macros that, in effect, represented higher level commands. You must do that; right? You're not going to write out each little thing every single time.

Steve: Well, we're going to talk - one of the things we're going to talk about in two weeks is the nature of indirection and pointers.

Leo: Oh, boy. That's fun.

Steve: And...

Leo: Oh, boy. If you - that was - there are two things I found very difficult to learn in programming. Indirection was one, and recursion was the other.

Steve: It's hard. It requires you being very clear about whether you mean something or the thing that that thing points to.

Leo: Right. I remember it very well. Now it's obvious to me. But I do remember very well when I first started writing in C, learning where to put that little caret and where not to. Oh, this'll be fun.

Steve: Yeah.

Leo: This'll be fun. Oh, I'm really enjoying this, Steve. And it's bringing back memories, and it makes me want to drag out my copy of MSM.

Steve: Well, and, I mean, what we just described, I mean, that is - what I described is a working computer that has everything it needs to get something done. And I think the mystery or the surprise is that just that, I mean, that's all our computers do. They load and store and perform simple operations on little bits of data. And, I mean, look what we get as a result. Because they're able - because there's enough of these little bits of data, and it's able to do stuff so fast, that they perform magic, really.

Leo: Very awesome. Steve Gibson, you da man. Thank you very much for this show and everything you do. If you are at all interested in more of Steve, you can get all the Steve you want at GRC.com. That's the Gibson Research Corporation, GRC.com. That's where SpinRite lives. You might as well just run over there and get one right now. You're going to need it someday. If you're got a hard drive, you need SpinRite, the world's best, frankly only decent hard drive recovery and maintenance utility, GRC.com.

While you're there, check out this show's notes, transcriptions, 16KB versions for the bandwidth-impaired. Steve's got great show notes. We have some visitors in the studio, Steve, and - what's your name? Alex was saying that as a student, a computer science student, he had an assignment to talk about web security. And he went to the show notes, and he got the transcriptions. And he says, "It's the only thing that really helped me understand it so I could write this paper."

Steve: Cool.

Leo: So you really provide a real service. GRC.com. You can watch this show, we do it live every Wednesday at 2:00 p.m. Pacific, 11:00 a.m. - I'm sorry, 2:00 p.m. Eastern, 11:00 a.m. Pacific, that's 1800 UTC at live.twit.tv. And of course iTunes and the Zune store and all the - Winamp, everybody has a subscription to this podcast. But you can find it directly at TWiT.tv/sn. There's a little subscription dropdown there, and you can pick your poison. GRC.com for Steve's site; TWiT.tv/sn for ours. Steve, we'll see you next week.

Steve: Talk to you then, Leo. Thanks.