CHAPTER 5

Firmware

In this chapter, you will learn how to

- Explain the function of BIOS
- Distinguish among various CMOS setup utility options
- Describe option ROM and device drivers
- Troubleshoot the power-on self test (POST)
- Maintain BIOS and CMOS properly

In Chapter 3, “CPUs,” you saw how the address bus and data bus connect RAM to the CPU via the memory controller to run programs and transfer data. Assuming you apply power in the right places, you don’t need anything else to make a simple computer. The only problem with such a simple computer is that it would bore you to death—there’s no way to do anything with it! A PC needs devices such as keyboards and mice to provide input, and output devices such as monitors and speakers to communicate the current state of the running programs to you. A computer also needs permanent storage devices, such as solid-state drives, to store programs and data when you turn off the computer.

This chapter discusses in detail the software that controls a PC at its core. We’ll start with a couple of sections on why and how it all works, and then we’ll look at hardware and self-testing circuits. The chapter finishes with the finer points of maintaining this essential programming and hardware.

1001

We Need to Talk

For a keyboard or a monitor or a hard drive to work with a CPU, they must communicate via some kind of physical connection. More than that, these peripherals (usually) can't connect directly to the CPU. This communication requires a controller, a chip that connects the device to the CPU (see Figure 5-1).

Figure 5-1 A controller chip acts as an interface

Getting the CPU to communicate with a controller starts with some kind of interconnection—a communication bus (that means *wires*) that enables the CPU to send commands to and from devices. To make this connection, let's extend the data bus and the address bus throughout the motherboard, connecting all of the computer's controllers to the CPU (see Figure 5-2).

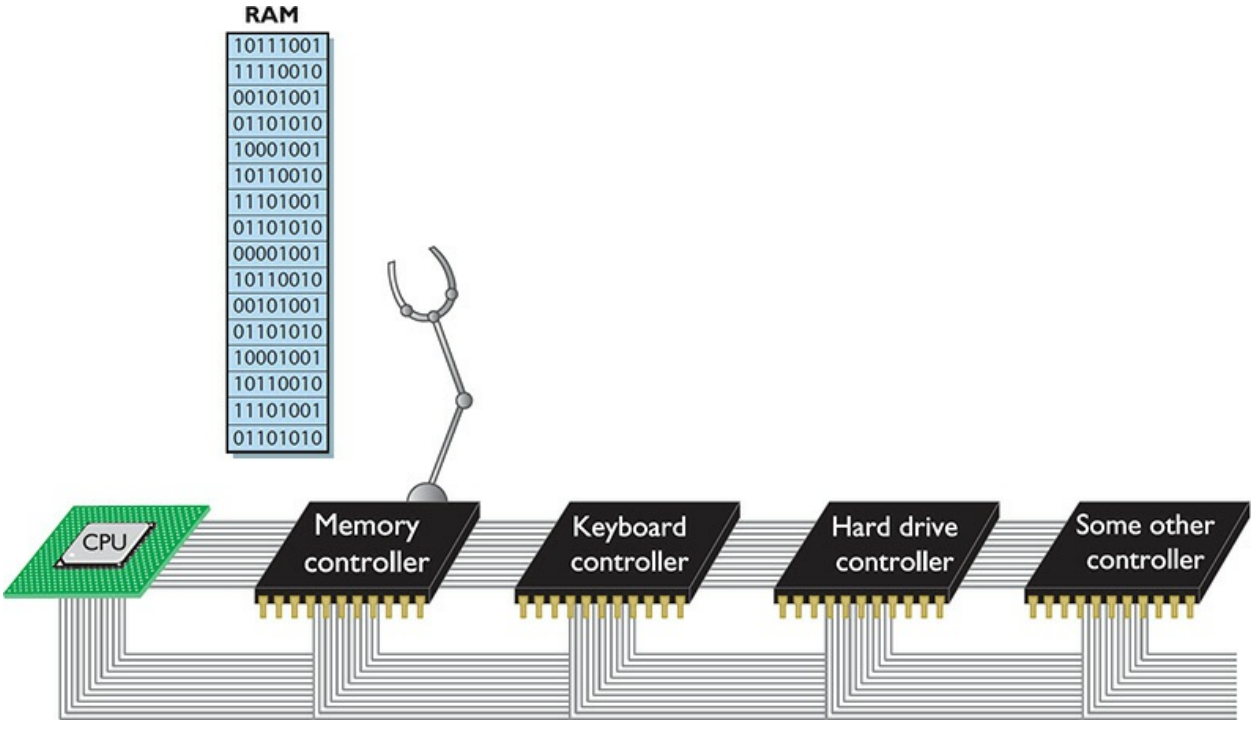

Figure 5-2 Data bus and address bus extended

Early motherboards were covered in controller chips. Figure 5-3 shows a very early motherboard, absolutely covered in controller chips (as well as many other chips).

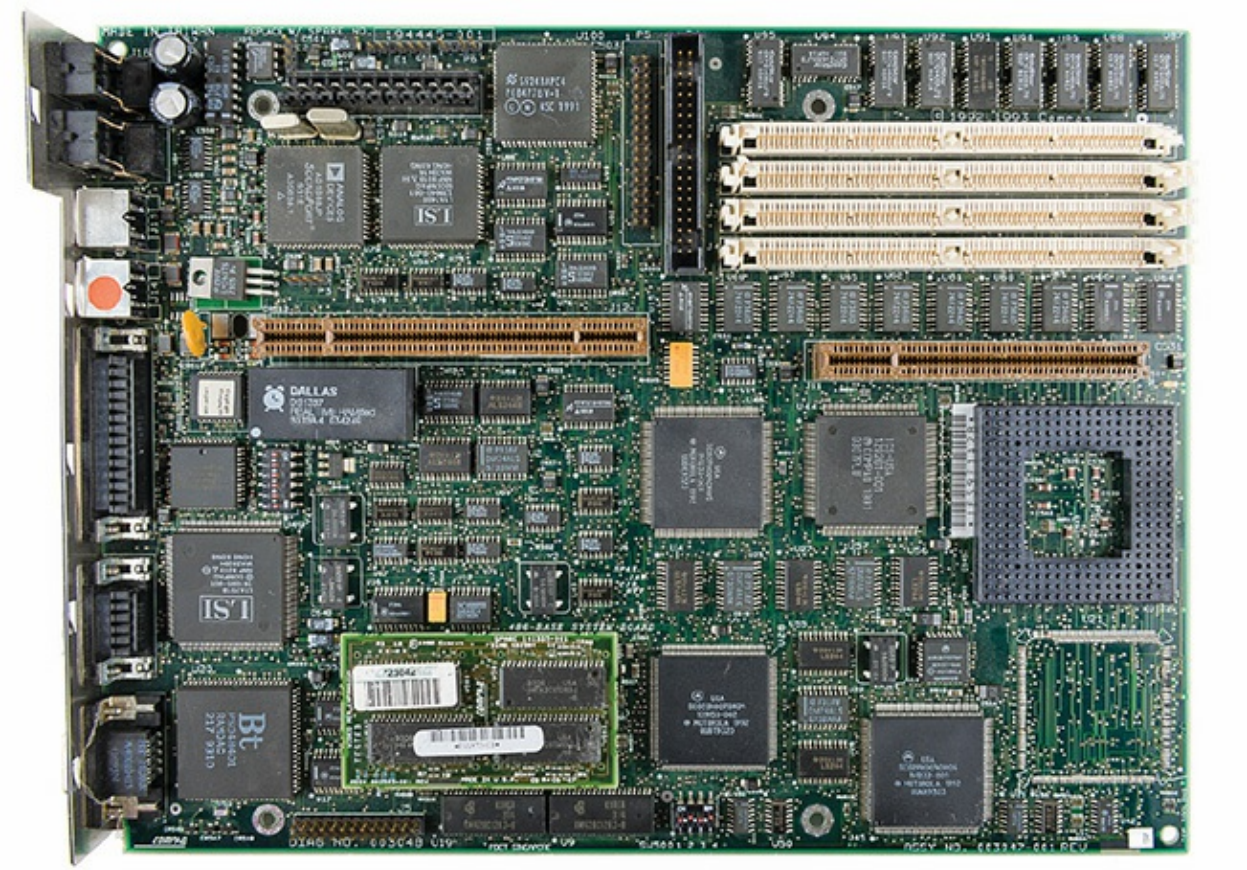

Figure 5-3 Early motherboard, loaded with chips

Over time, chip manufacturers began to combine multiple controllers into specifically designed *chipsets*. Early chipsets such as the Intel 430VX shown in Figure 5-4 consisted of two paired chips called the *northbridge* and *southbridge*.

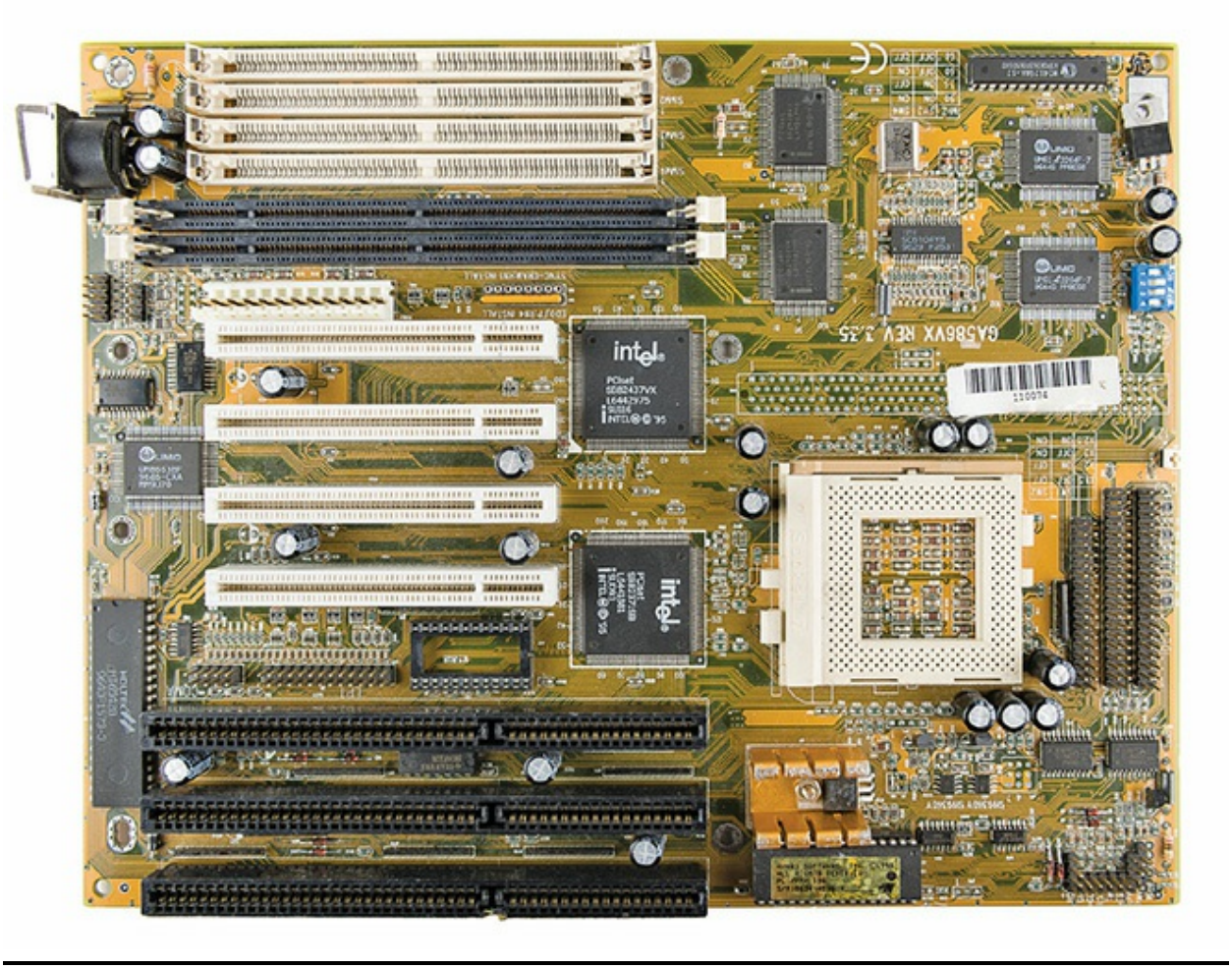

Figure 5-4 Ancient 430VX chipset showing northbridge and southbridge

Chipsets were in pairs for many years, roughly from 1990 to around 2010. Today's CPUs have controllers built in, such as the memory and display controllers. Almost all chipsets are now a single chip–Intel's name for this chip is *Platform Controller Hub* (*PCH*). Figure 5-5 shows a motherboard with both the CPU and the PCH visible. AMD (and most of the tech industry) refers to the chip as the chipset.

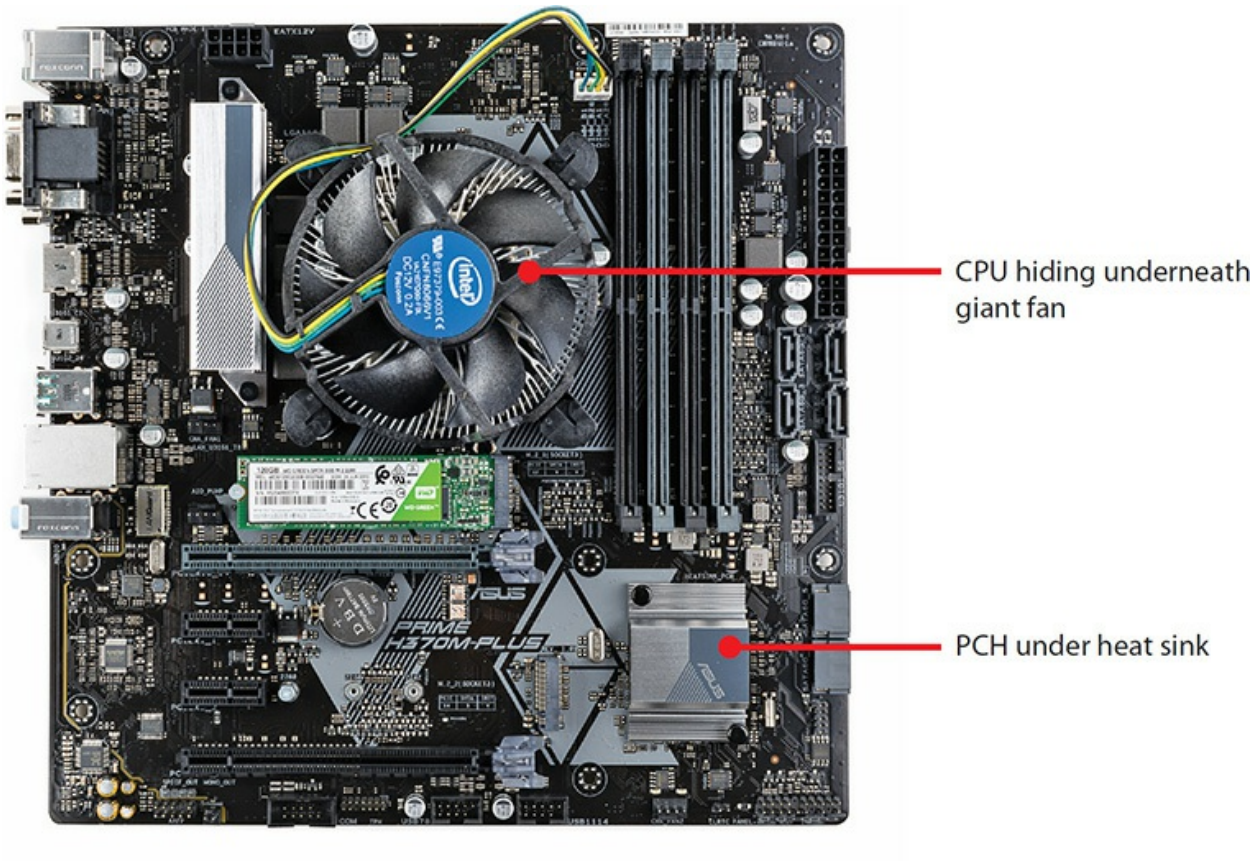

Figure 5-5 Intel CPU and PCH

NOTE All but the lowest-powered chipsets require cooling fins or fans. Also, even though chipsets are often single-chip solutions, it's common tech speak to say "chipset" (even though "set" implies more than one chip).

The chipset extends the data bus to every device on the PC. The CPU uses the data bus to move data to and from all the devices of the PC. Data constantly flows on the data bus among the CPU, chipset, RAM, and other devices on the PC (see Figure 5-6).

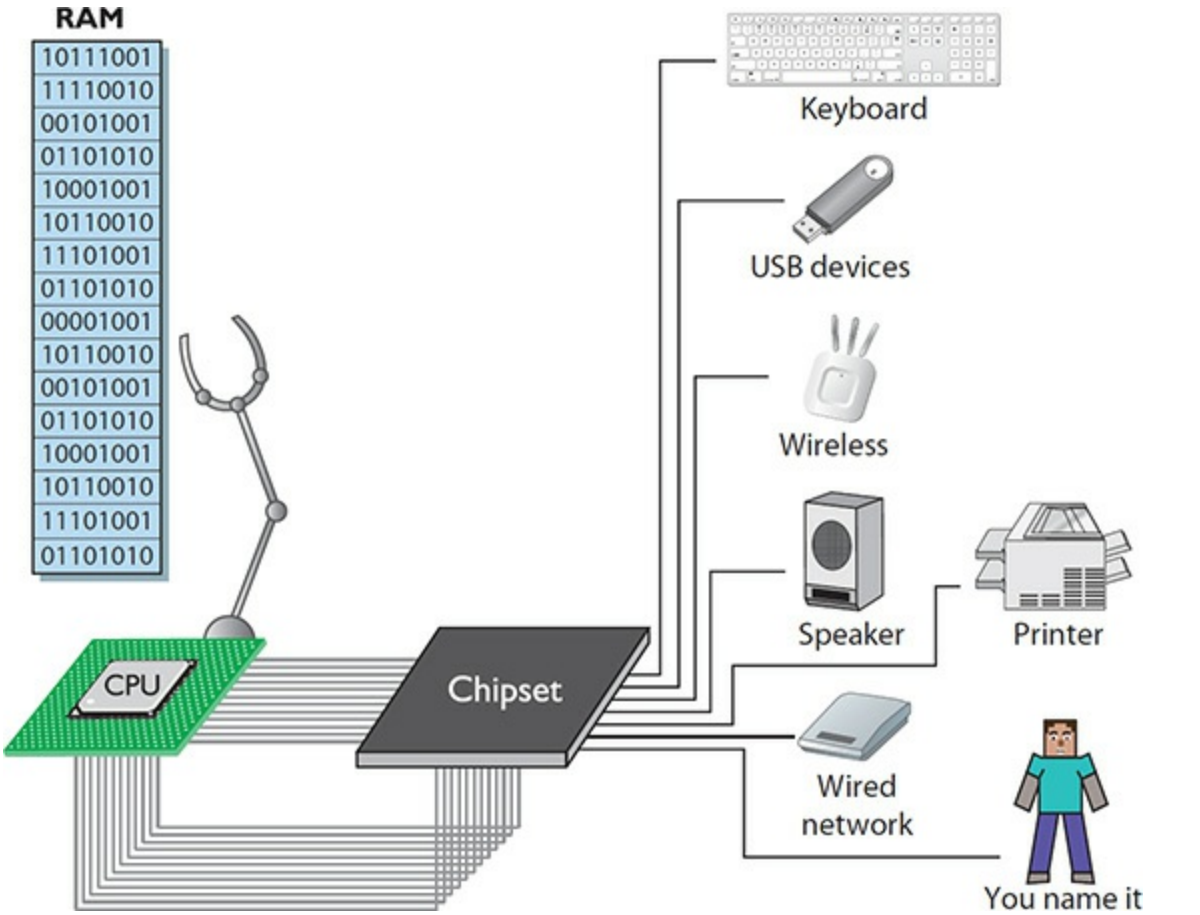

Figure 5-6 Everything connecting

It’s not too hard to swallow the concept that the CPU uses the address bus to talk to the devices, but how does it know what to *say* to them? How does it know all of the patterns of ones and zeros to place on the address bus to tell the hard drive it needs to send a file? Let’s look at the interaction between the keyboard and CPU for insight into this process.

Talking to the Keyboard

The keyboard provides a great example of how the buses and support programming help the CPU get the job done. In early computers, the keyboard connected to the data bus via a special chip known as the *keyboard controller*. Don’t bother looking for this chip on your motherboard—the chipset handles keyboard controller functions. The way the keyboard controller—or technically, the keyboard controller *circuitry*—works with the CPU, however, has changed only a small amount in the past decades, making

it a perfect tool to illustrate how the CPU talks to a device.

NOTE Techs commonly talk about various functions of the chipset as if those functions were still handled by discrete chips. You'll hear about memory controllers, keyboard controllers, mouse controllers, USB controllers, and so on, even though they're all just circuits on the CPU or chipset.

The keyboard controller was one of the last single-function chips to be absorbed into the chipset. Figure 5-7 shows a typical keyboard controller from those days. Electronically, it looked like Figure 5-8.

Figure 5-7 A keyboard chip on an older motherboard

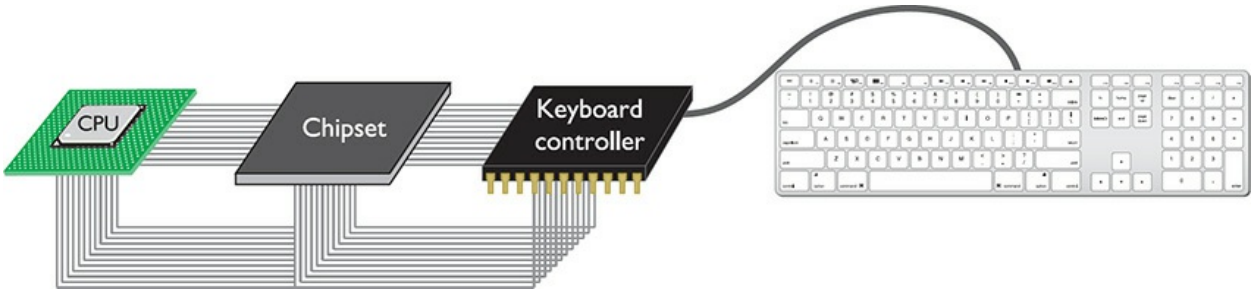

Figure 5-8 Electronic view of the keyboard controller

Every time you press a key on your keyboard, a scanning chip in the keyboard notices which key you pressed. Then the scanner sends a coded pattern of ones and zeros—called the *scan code*—to the keyboard controller. Every key on your keyboard has a unique scan code. The keyboard controller stores the scan code in its own register. Does it surprise you that the lowly keyboard controller has a register similar to a CPU? Lots of chips have registers—not just CPUs (see Figure 5-9).

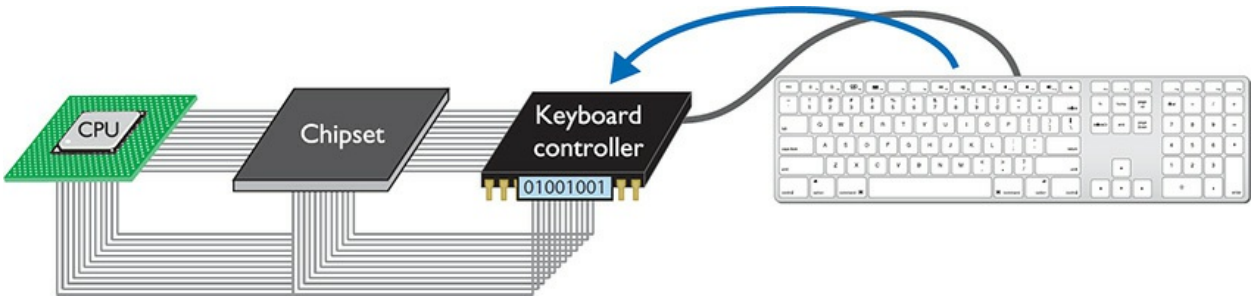

Figure 5-9 Scan code stored in keyboard controller's register

How does the CPU get the scan code out of the keyboard controller? While we're at it, how does the CPU tell the keyboard to change the typematic buffer rate (when you hold down a key and the letter repeats) or to turn the number lock LED on and off, to mention just a few other jobs the keyboard needs to do for the system? The point is that the keyboard controller must be able to respond to multiple commands, not just one.

The keyboard controller accepts commands exactly as you saw the CPU accept commands in Chapter 3. Remember when you added 2 to 3 with the 8088? You had to use specific commands from the 8088's codebook to tell the CPU to do the addition and then place the answer on the external data bus. The keyboard controller has its own codebook—much simpler than any

CPU's codebook, but conceptually the same. If the CPU wants to know what key was last pressed on the keyboard, the CPU needs to know the command (or series of commands) that orders the keyboard controller to put the scan code of the letter on the external data bus so the CPU can read it.

BIOS

The CPU doesn't magically or otherwise automatically know how to talk with any device; it needs some sort of support programming loaded into memory that teaches it about a particular device. This programming is called *basic input/output services* (*BIOS*). The programs dedicated to enabling the CPU to communicate with devices are called *services* (or device drivers, as you'll see later in the chapter). This goes well beyond the keyboard, by the way. In fact, *every* device on the computer needs BIOS! But let's continue with the keyboard for now.

Bringing BIOS to the PC

A talented programmer could write BIOS for a keyboard if the programmer knew the keyboard's codebook; keyboards are pretty simple devices. This begs the question: Where would this support programming be stored? Programming could be incorporated into the operating system. Storing programming to talk to the hardware of your PC in the operating system is great—all operating systems have built-in code that knows how to talk to your keyboard, your mouse, and just about every piece of hardware you may put into your PC.

That's fine once the operating system's up and running, but what about a brand-new stack of parts you're about to assemble into a new PC? When a new system is being built, it has no operating system. The CPU must have access to BIOS for the most important hardware on your PC: not only the keyboard, but also the monitor, mass storage drives, optical drives, USB ports, and RAM. This code can't be stored on a hard drive or optical disc—these important devices need to be ready at any time the CPU calls them, even before installing a mass storage device or an operating system.

The perfect place to store the support programming is on the motherboard. That settles one issue, but another looms: What storage medium should the motherboard use? DRAM won't work, because all of the data would be

erased every time you turned off the computer. You need some type of permanent program storage device that does not depend on other peripherals to work. And you need that storage device to sit on the motherboard.

ROM

Motherboards store the keyboard controller support programming, among other programs, on a special type of device called a *read-only memory* (*ROM*) chip. A ROM chip stores programs, *services*, exactly like RAM. ROM differs from RAM in two important ways. First, ROM chips are *nonvolatile*, meaning that the information stored on ROM isn't erased when the computer is turned off. Second, traditional ROM chips are read-only, meaning that once you store a program on one, you can't change it.

Motherboards use a type of ROM called *flash ROM*, the same stuff that stores your data in your smartphone or SSD. That is why we call updating the BIOS firmware, "flashing the BIOS," which we will cover later in this chapter. Figure 5-10 shows a typical flash ROM chip on a motherboard.

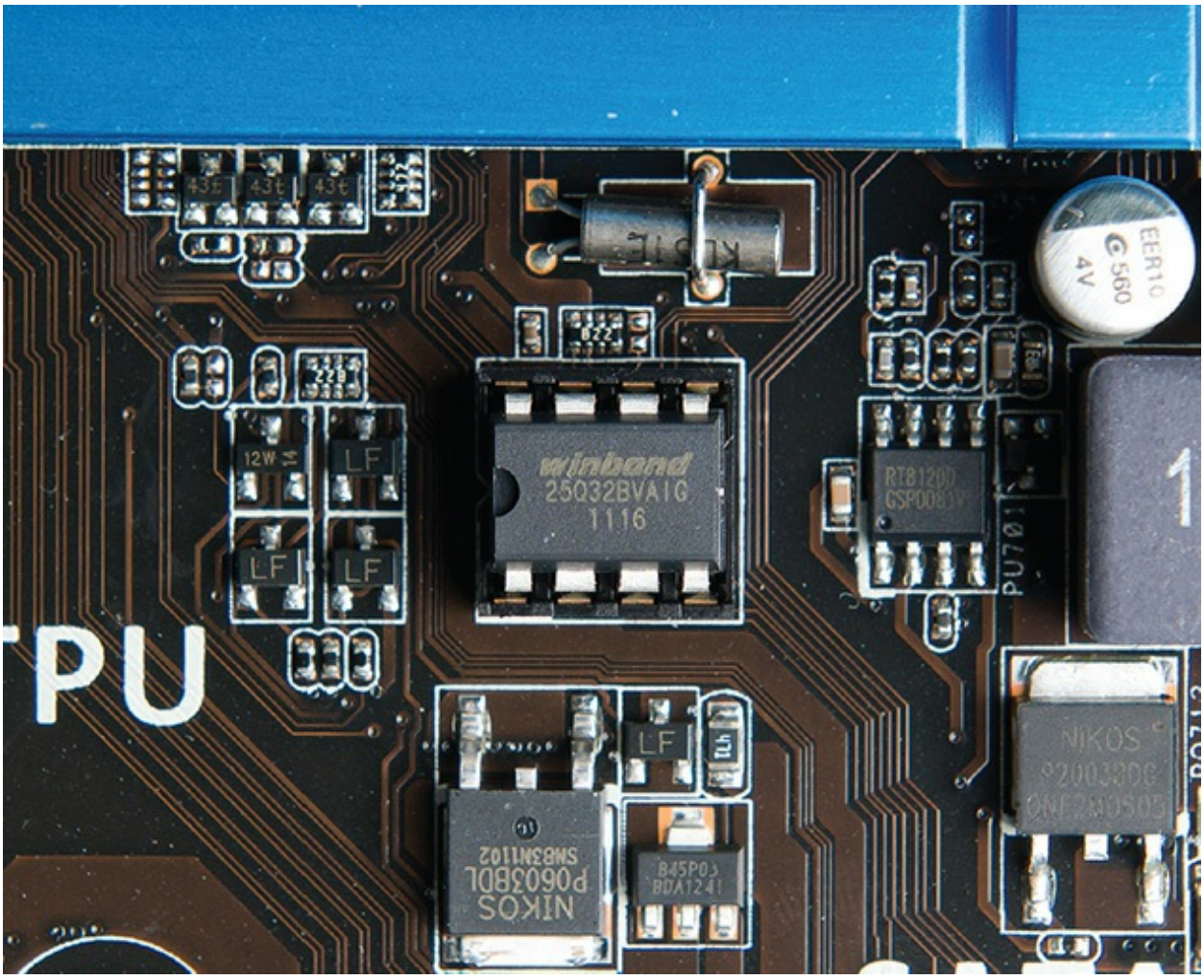

Figure 5-10 Typical flash ROM

Every motherboard has a flash ROM chip, called the *system ROM* chip because it contains code that enables your CPU to talk to the basic hardware of your PC (see Figure 5-11). As alluded to earlier, the system ROM holds BIOS for more than just the keyboard controller. It also stores programs for communicating with hard drives, optical drives, display devices, USB ports, and other basic devices on your motherboard.

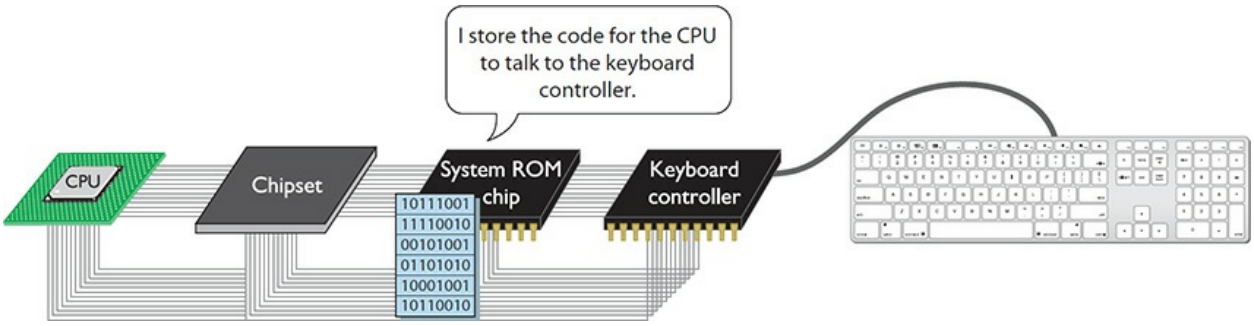

Figure 5-11 Function of the flash ROM chip

To talk to all of that hardware requires hundreds of little services (2 to 30 lines of code each). These hundreds of little programs stored on the system ROM chip on the motherboard are called, collectively, the *system BIOS* (see Figure 5-12). Techs call programs stored on ROM chips of any sort *firmware*. You will commonly hear techs today refer to the *system firmware*—what CompTIA (and this book) calls system BIOS.

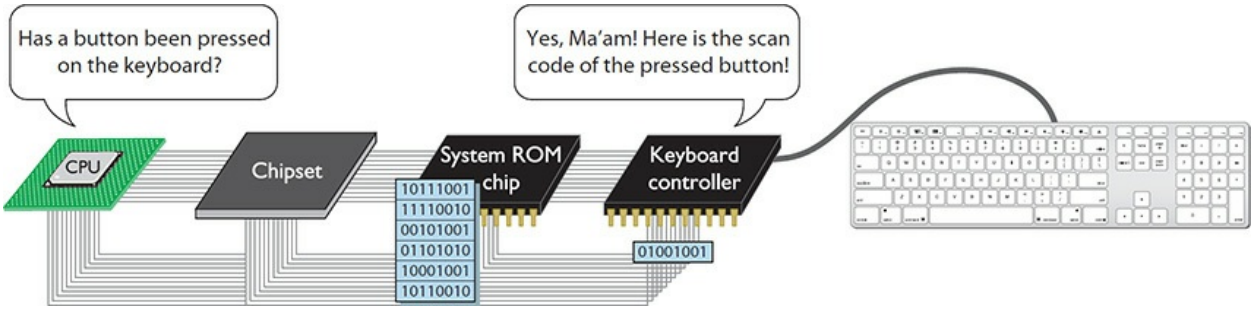

Figure 5-12 CPU running BIOS service

EXAM TIP Programs stored on ROM chips—flash or any other kind of ROM chip—are known collectively as *firmware*, as opposed to programs stored on dynamic media, which are collectively called *software*.

System BIOS Support

Every system BIOS has two types of hardware to support. First, the system BIOS supports all of the hardware that never changes, such as the keyboard.

Another example of hardware that never changes is the PC speaker (the tiny one that beeps at you, not the ones that play music). The system ROM chip stores the BIOS for these and other devices that never change.

Second, the system BIOS supports all of the hardware that might change from time to time. This includes RAM (you can add RAM) and hard drives (you can replace your hard disk drive [HDD] with a larger drive or a solid-state drive [SSD] or add a second drive of either type). The system ROM chip stores the *BIOS* for these devices, but the system needs another place to store information about the specific *details* of a piece of hardware. This enables the system to differentiate between a Western Digital Blue 4-TB HDD and a Samsung 860 EVO 2-TB SSD, and yet still support both drives right out of the box.

UEFI

Modern systems use firmware programming called the *Unified Extensible Firmware Interface* (*UEFI*). Here are a few advantages of UEFI over the original BIOS in PCs:

- UEFI supports booting to partitions larger than 2.2 TB.
- UEFI firmware is native 32- or 64-bit; this lets the manufacturers include lots of features for setup and diagnoses.
- UEFI handles all boot-loading duties; no more jumping from boot sector to boot sector.
- UEFI is portable to other chip types, not just 16-bit x86.

All current systems use UEFI. Many also provide legacy support for BIOS services in case you feel some retro gaming is in order. But a zillion older systems use the older BIOS. Most techs continue to call the support software BIOS, even though technically the terms differ. There's no standardization on how to pronounce UEFI, by the way. Microsoft initializes it: "U-E-F-I." Others say "you-fee" or "you-fie." For a job interview, stick with initializing it. You can't go wrong that way.

CMOS and RTC

Because the BIOS firmware is stored in ROM, and ROM is *read only*, it

needs a place to store all its settings so they don't have to be re-entered every time you boot your computer. That place is a tiny bit of RAM hooked up to a small battery to keep it working with the PC off. We call this memory the *complementary metal-oxide semiconductor* (*CMOS*) chip (Figure 5-13). In addition to storing all the various BIOS settings, the CMOS also handles the system's *real-time clock* (*RTC*) so you don't have to keep setting the time on every boot.

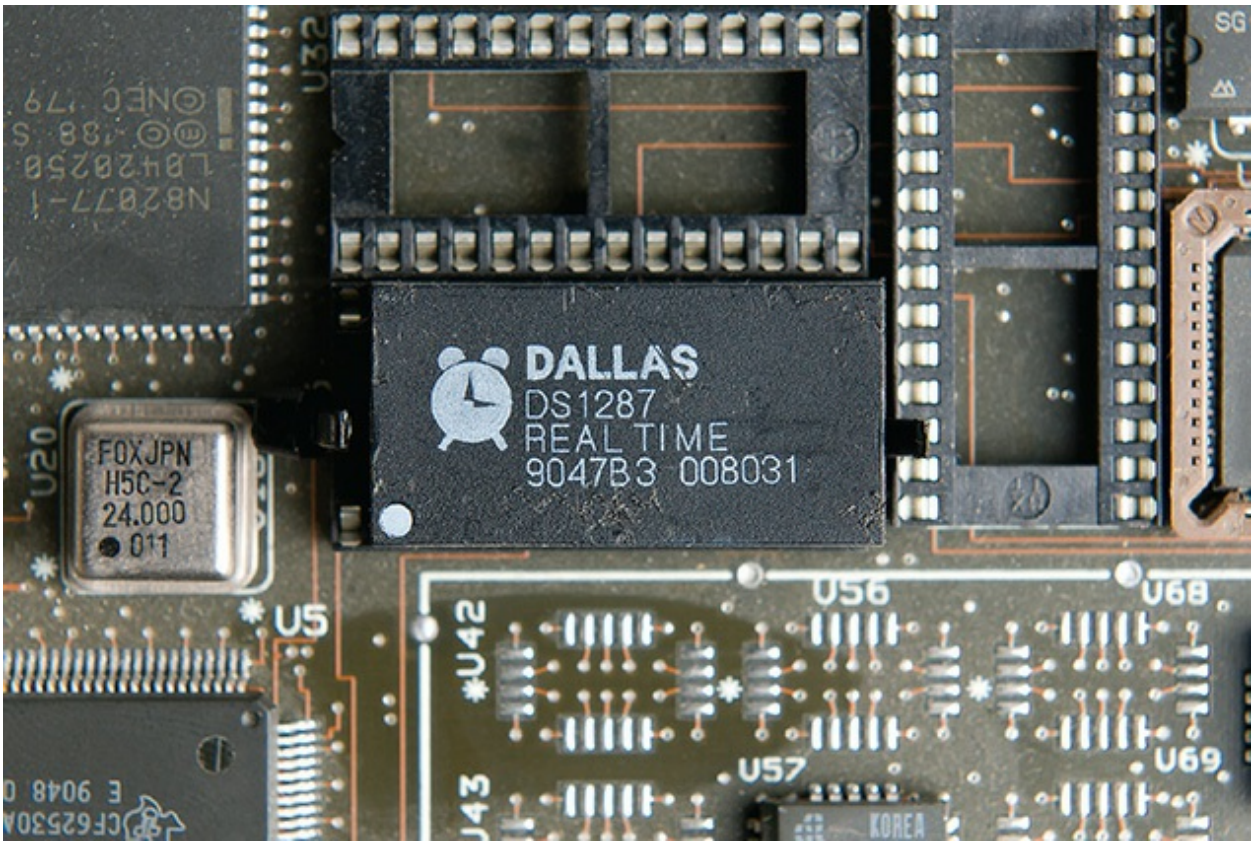

Figure 5-13 Old-style CMOS

The standalone CMOS chip has long since been incorporated into the main chipset. The information stored in CMOS is necessary for the PC to function.

If the data stored in CMOS about a piece of hardware (or about its fancier features) is different from the specs of the actual hardware, the computer cannot access that piece of hardware (or use its fancier features). It is crucial that this information be correct. If you change any of the previously mentioned hardware, you must update CMOS to reflect those changes. You

need to know, therefore, how to change the data in CMOS.

EXAM TIP All the details of UEFI and CMOS that you're going to spend many hours memorizing for the 1001 exam only apply to PCs and Linux machines. Apple computers have EFI and CMOS, but Apple designs their systems from the ground up as unified systems. Apple has done all the work for you and you simply use the macOS machine.

Every PC ships with a program built into the system ROM called the *system setup utility* that enables you to access and modify CMOS data. On a completely new system, with no operating system installed, you'll see something like Figure 5-14. After the OS is installed, these screens effectively disappear. I'll show you how to access them in a little bit.

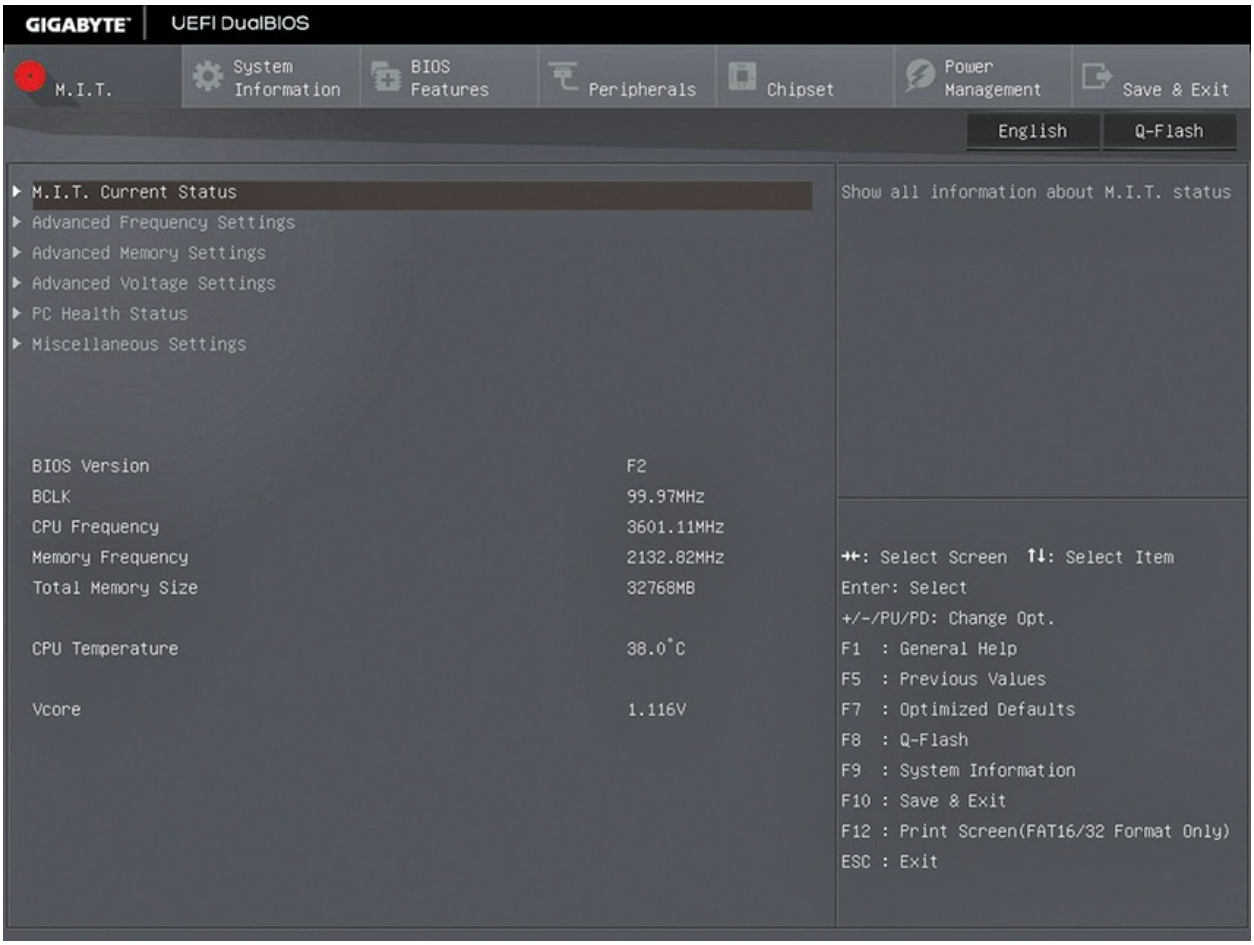

Figure 5-14 Gigabyte system setup utility

NOTE The terms *CMOS setup program*, *CMOS*, and *system setup utility* are functionally interchangeable today. You'll even hear the program referred to as the *BIOS setup utility*, UEFI/BIOS setup, or UEFI firmware settings.

Typical System Setup Utility

Every BIOS/UEFI maker's system setup utility looks a little different, but don't let that confuse you. They all contain basically the same settings; you just have to be comfortable poking around the different interfaces. To avoid

doing something foolish, *do not save anything* unless you are sure you have it set correctly.

Several years ago, BIOS/UEFI manufacturers in the consumer space migrated to graphical system setup utilities that enable you to use a mouse. You'll still find plenty of examples in the field of the classic text-only system setup utilities. You need to know both, so this section will show you both styles. We'll run through a graphical version first, then skim through an older text-only version.

Graphical UEFI System Setup Utility

Figure 5-15 shows a typical, simple graphical setup screen. This system setup utility has two modes: EZ and Advanced. You can't do much in this first EZ Mode screen except view information about installed components, select one of three preset System Performance optimization options, and change the boot priority.

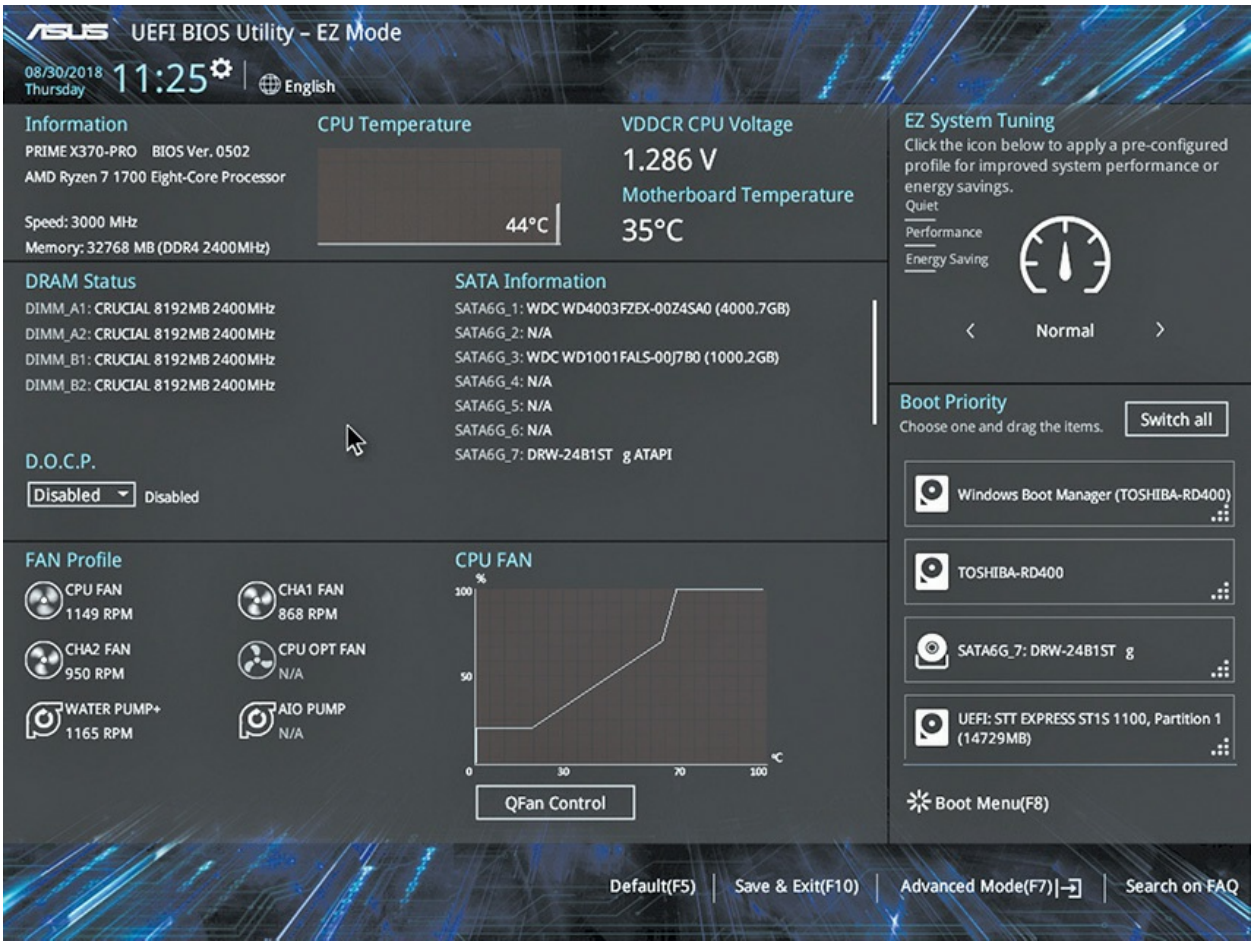

Figure 5-15 ASUS UEFI system setup utility

Click the option to go into Advanced Mode and you'll get a much more versatile utility (see Figure 5-16) for changing the interface configurations. The Main tab offers some *BIOS component information*, such as surface details on amount of RAM and speed of CPU, plus a couple of options to modify the language and date and time. (Some utilities will show information about installed hard drives and optical drives; this UEFI firmware presents that information elsewhere.)

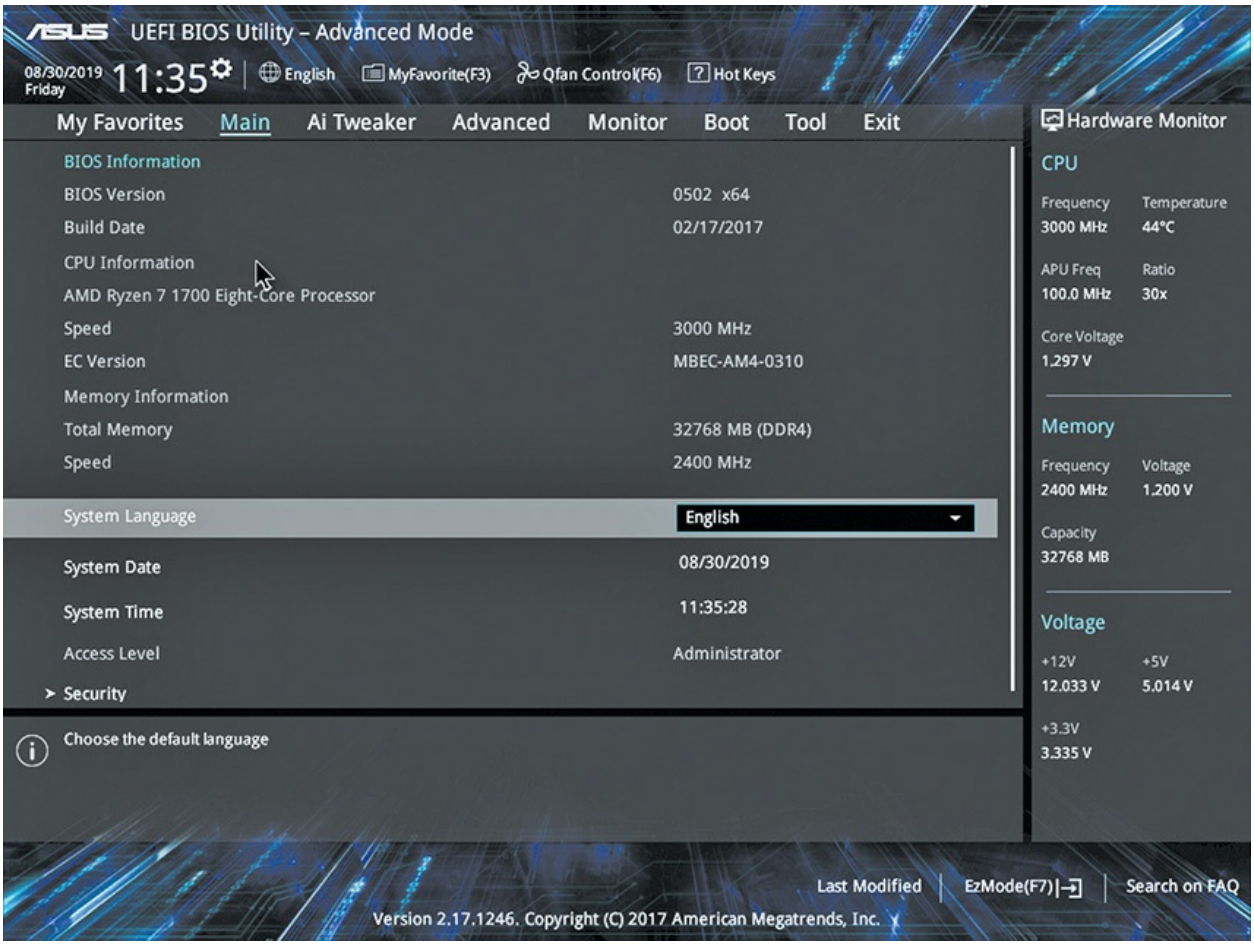

Figure 5-16 Main tab

The Main tab also enables you to configure modest *BIOS security* by setting an administrator or user password. (The default for the pictured UEFI BIOS is Access Level: Administrator. Click the Security option to change access information. UEFI setup screens differ somewhat, but you'll find

similar options in all of them.)

An *administrator password* locks or unlocks access to the system setup utility. A *user password* locks or unlocks the computer booting to an operating system. Set a BIOS/UEFI password when you encounter a scenario like installing computer kiosks at a convention or installing systems in a public library. A BIOS/UEFI password stops casual miscreants from messing with your accessible systems.

Things get far more interesting in the other tabs. Selecting the Ai Tweaker tab, for example, enables you to delve into the Dark Arts of overclocking both the CPU and RAM (see Figure 5-17). You can change the clock multiplier, clock speeds, voltages, and more here. This is a great place to go to fry a new CPU!

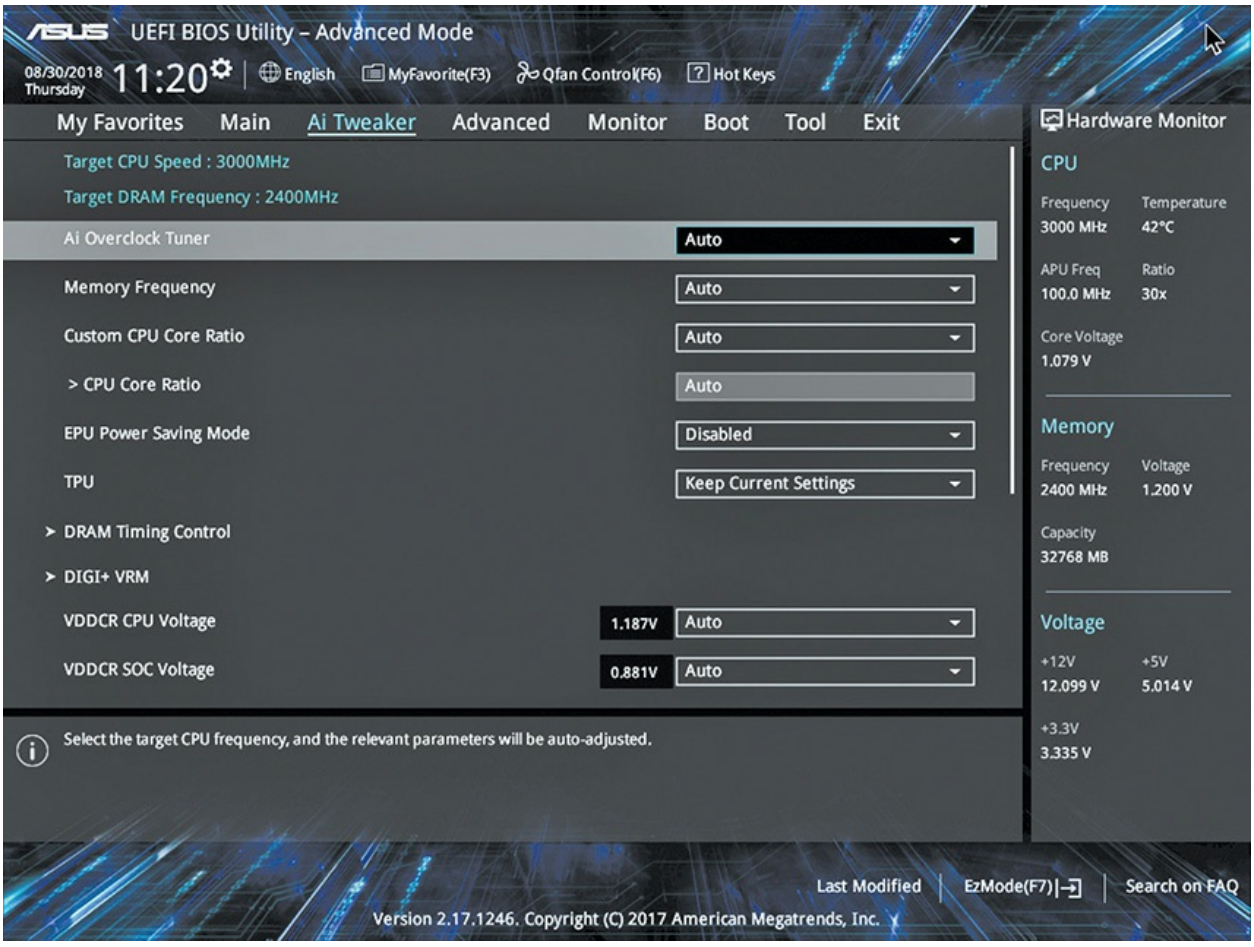

Figure 5-17 Ai Tweaker tab

The Advanced tab (see Figure 5-18) gives component information about

CPUs, hard drives and optical drives, and all the built-in components, such as USB ports. In this tab, as you drill down to each subcategory, you can configure drive settings, enable and disable devices, and more.

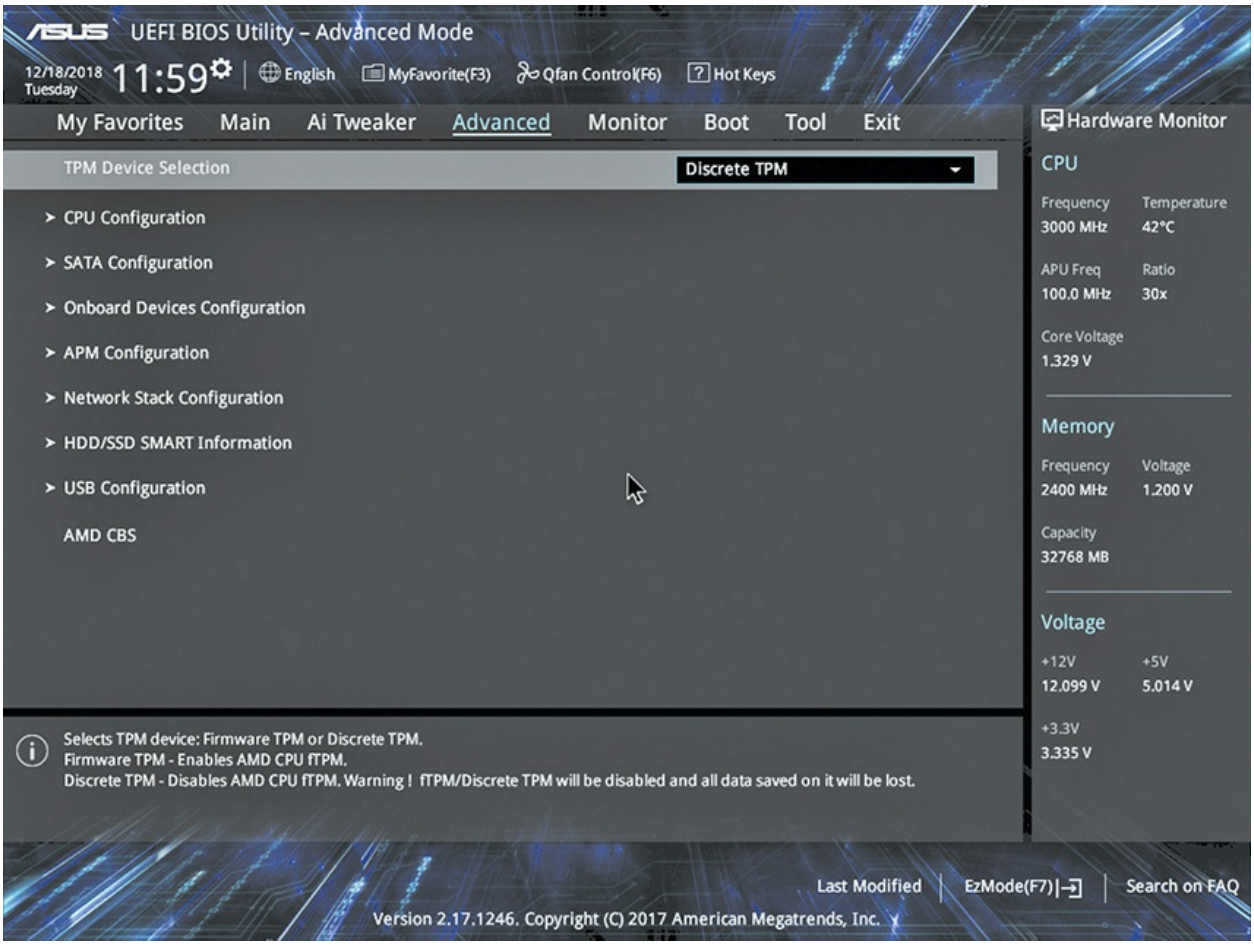

Figure 5-18 Advanced tab

The Monitor tab (see Figure 5-19) shows monitoring information for CPU and motherboard temperatures, fan speeds, and voltages. You can modify the behavior of the chassis fans here too. All of this monitoring information is considered some of the *built-in diagnostics* for both the motherboard and the full system.

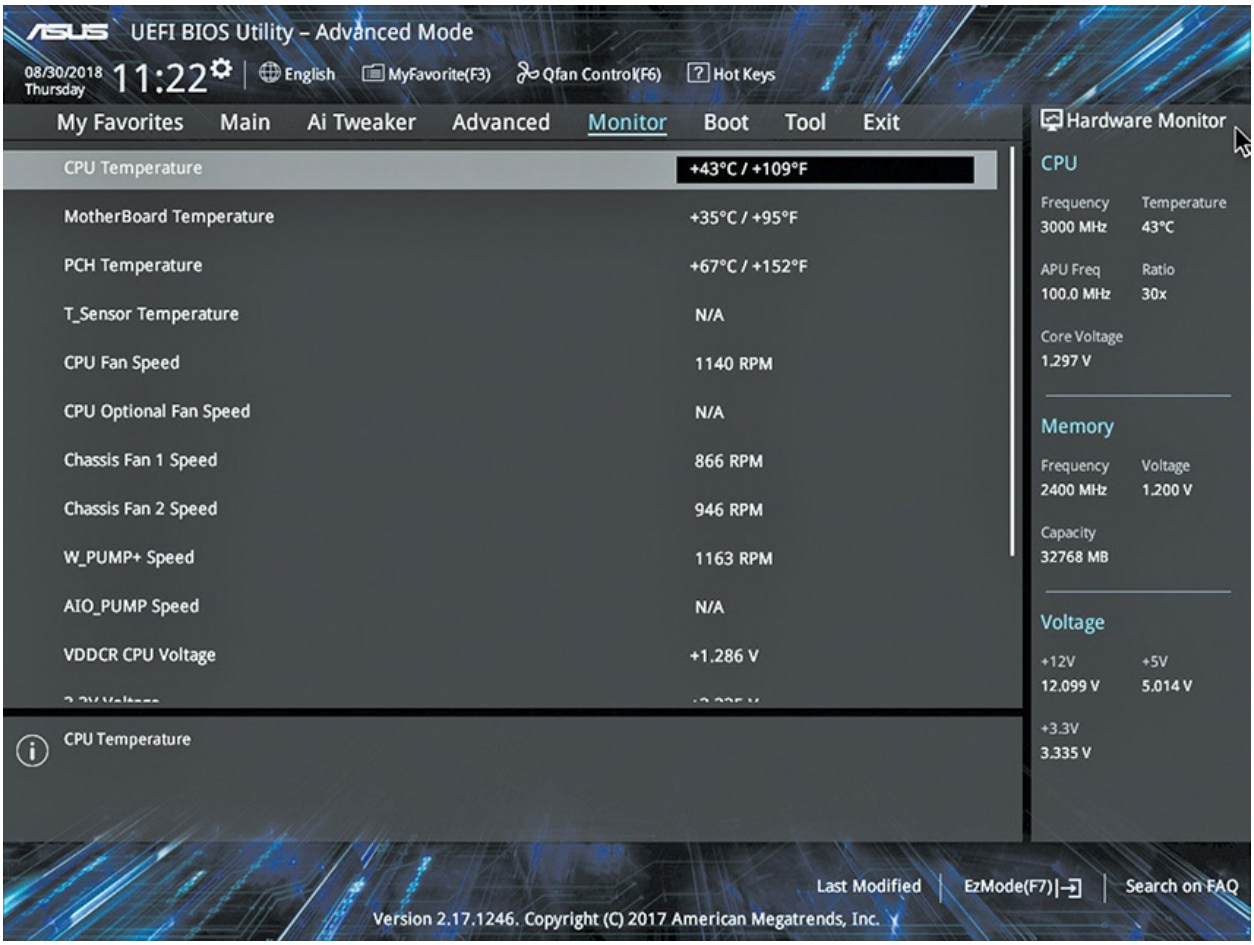

Figure 5-19 Monitor tab

NOTE Some systems refer to the Monitor display as PC Health.

The Boot tab (see Figure 5-20) enables you to adjust boot settings. You can select devices to boot by priority, setting the *boot sequence* used by the motherboard. (See “The Boot Process” later in this chapter for more information.) You can determine how the system will react/inform if booting fails, and more.

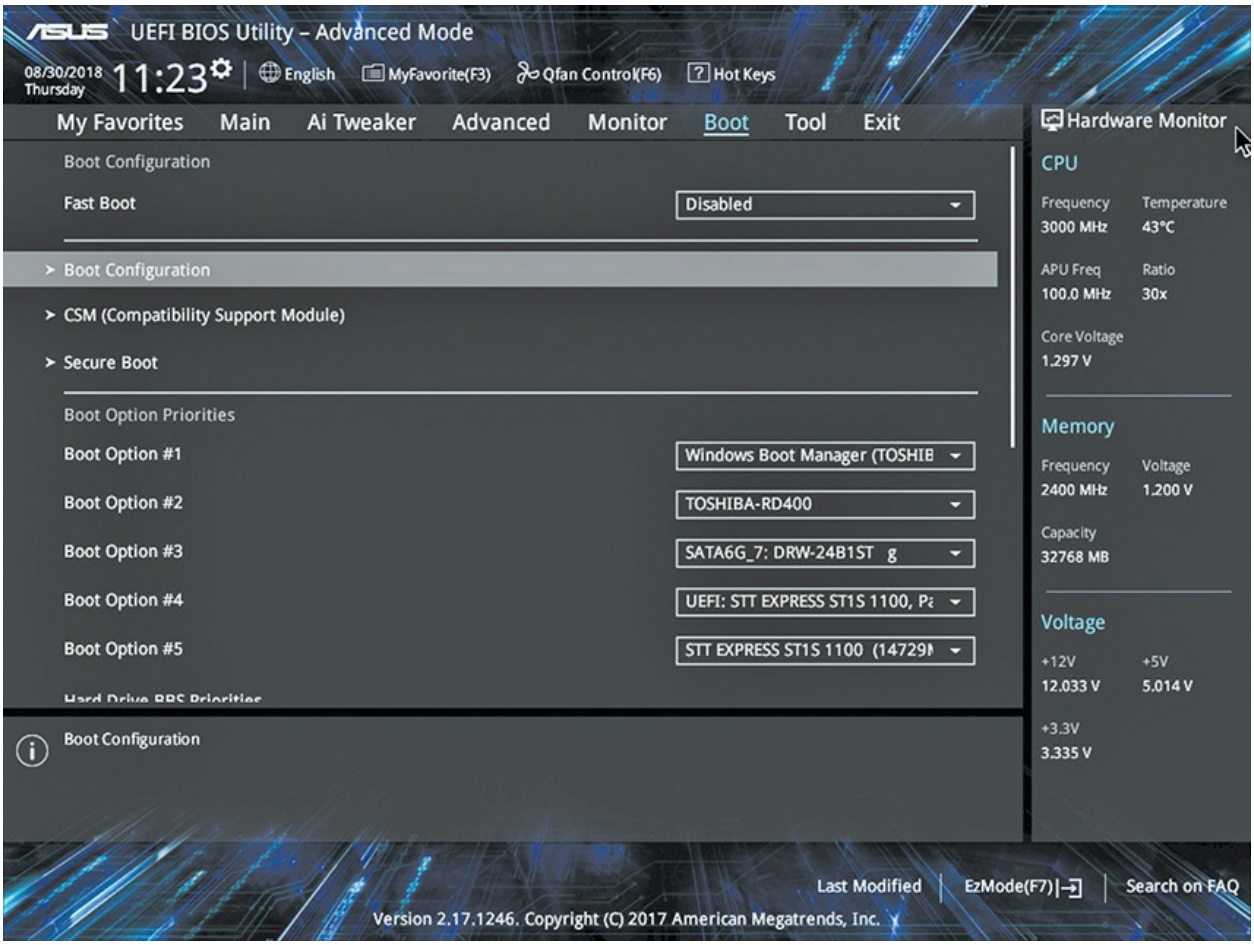

Figure 5-20 Boot tab

The Tool tab (see Figure 5-21) has a couple of very important features. The EZ Flash 3 utility enables you to update the motherboard firmware. See the "Flashing the ROM" section later in this chapter for more details. The Tool tab also shows RAM information. That's the SPD option (for *serial presence detect*) you should recognize from Chapter 4, "RAM."

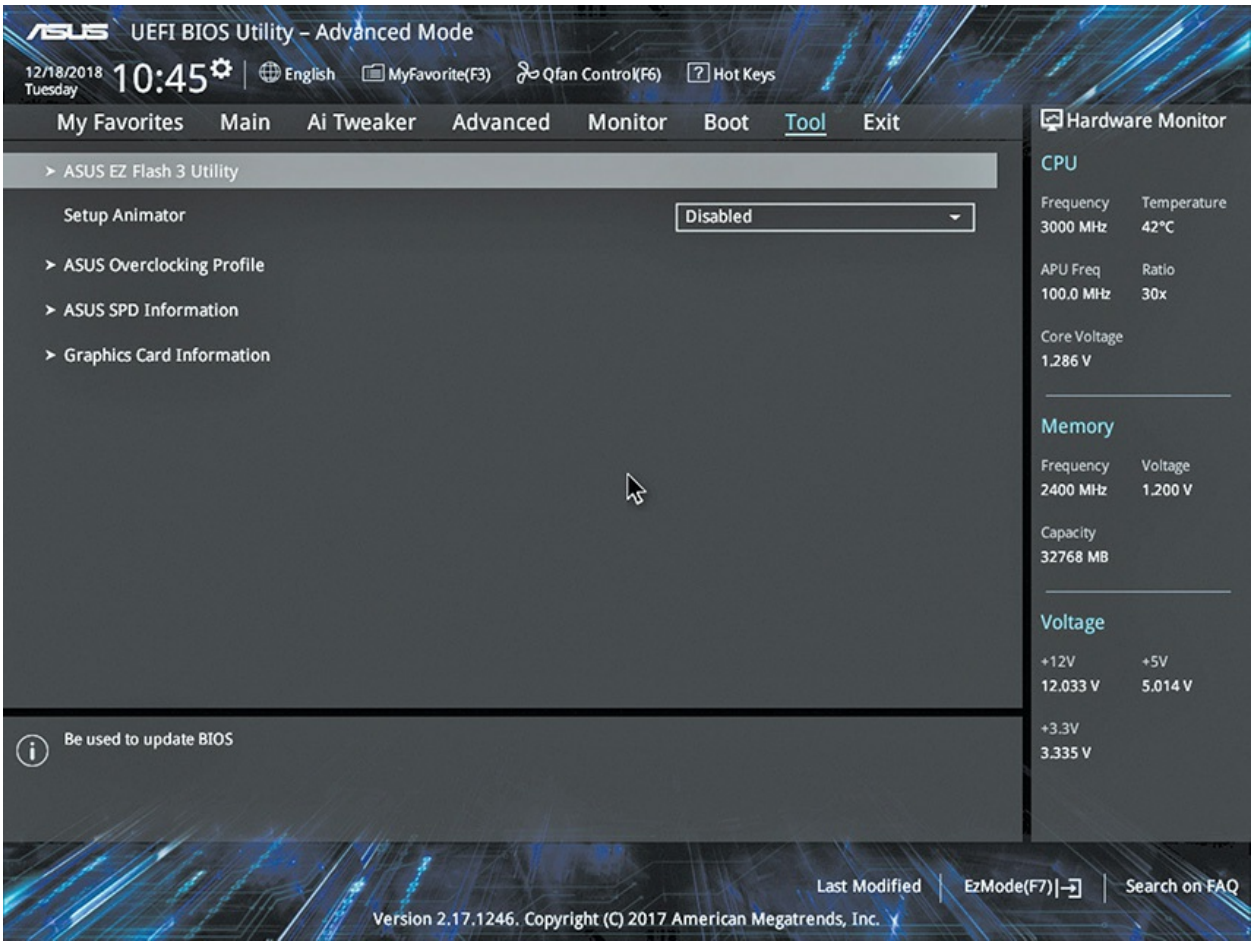

Figure 5-21 Tool tab

Text-Based UEFI Intel-Based Setup Utility

In this second walkthrough, we'll switch to a UEFI motherboard on an Intel-based portable computer. As we go through the screens, pay attention to the options listed on each. I'll call out features that the graphical AMD-based UEFI didn't have.

The Information tab (see Figure 5-22) offers straightforward information about the CPU and RAM amount, and cryptic information about the hard drive. Other tabs do more.

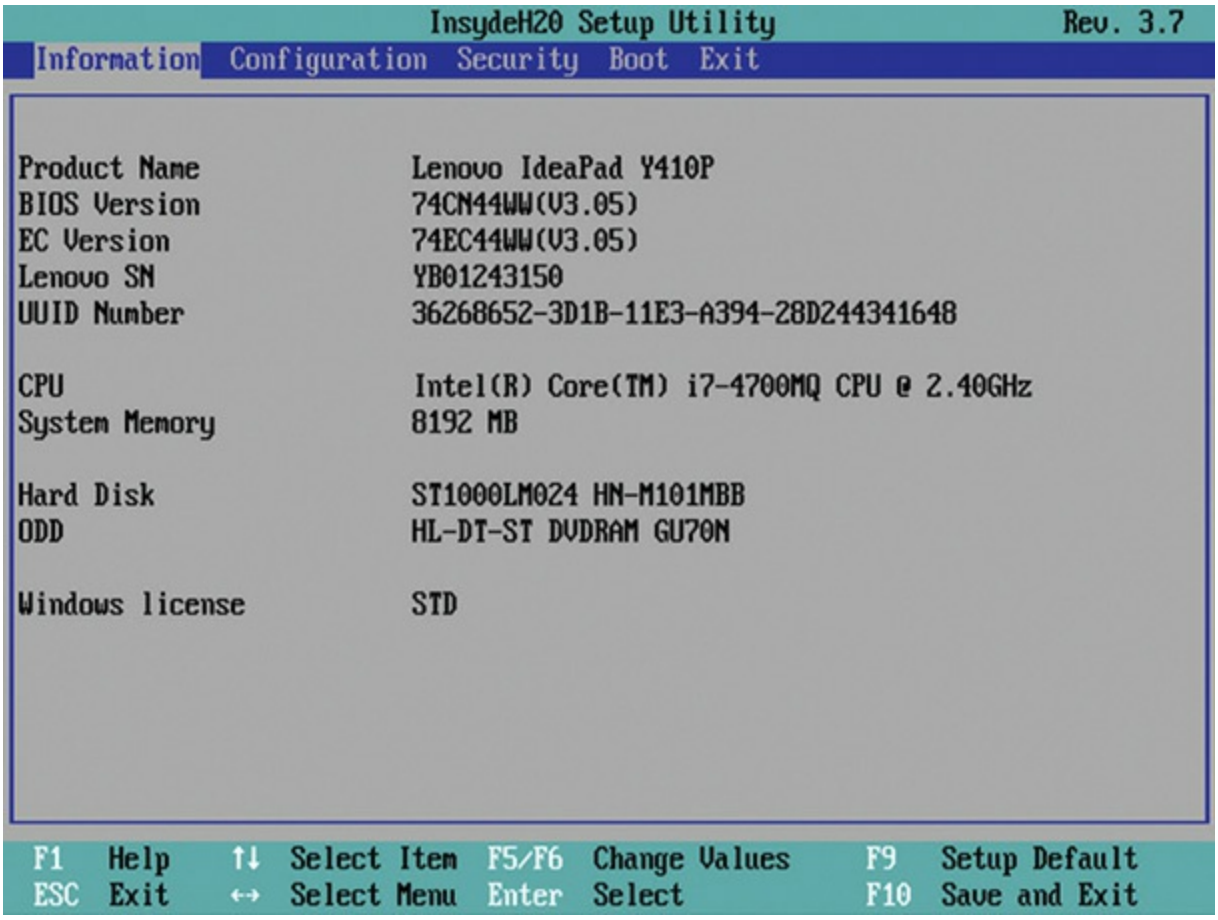

Figure 5-22 Information tab

The Configuration tab (see Figure 5-23) shows a number of built-in devices that you configure or enable/disable here. Because this is a portable, it has an option to turn on/off wireless networking capabilities.

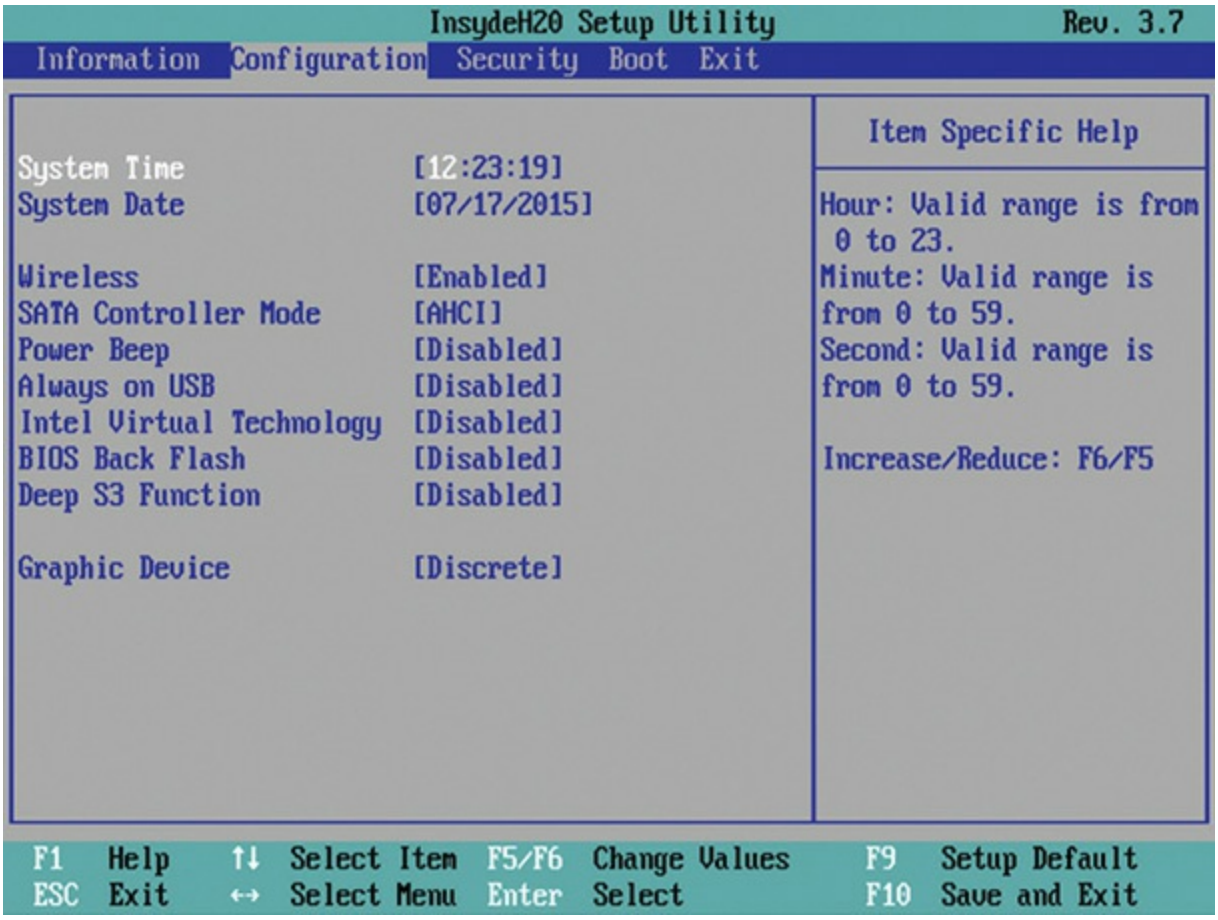

Figure 5-23 Configuration tab

There are two interesting options here that are covered in detail in other chapters but warrant a brief discussion now. The Intel Virtual Technology option enables or disables *virtualization support* for virtual machines.

A *virtual machine* is a powerful type of program that enables you to run a second (or third or fourth), software-based machine inside your physical PC. It re-creates the motherboard, hard drives, RAM, network adapters, and more, and is just as powerful as a real PC. To run these virtual machines, however, you'll need a very powerful PC—you are trying to run multiple PCs at the same time, after all.

To support this, CPU manufacturers have added *hardware-assisted virtualization*. Intel calls their version Intel Virtualization Technology (Intel VT for short), and AMD calls theirs AMD Virtualization (AMD-V) technology. This technology helps the virtual machines use your hardware more efficiently and is controlled by the BIOS. This feature is disabled by

default in BIOS, so if your virtual machine requires hardware-assisted virtualization, you'll need to enable it here.

NOTE Chapter 22, "Virtualization," covers virtual machines in gory detail. Stay tuned!

This particular laptop has built-in graphics courtesy of the Intel Core i7 processor, plus it has a dedicated add-on video card for gaming. The Graphic Device option, set here to Discrete, means to use the dedicated video card when possible. This uses more electricity than the graphics support using only the processor, but it makes for way better gaming!

NOTE Chapter 17, "Display Technologies," goes into video options (and gaming) in modern systems.

The Security tab (see Figure 5-24) offers a lot more options for configuring BIOS security than found on the Main tab of the AMD-based system. You see the Administrator Password and User Password options, but there's also an option to set a couple of different hard drive passwords.

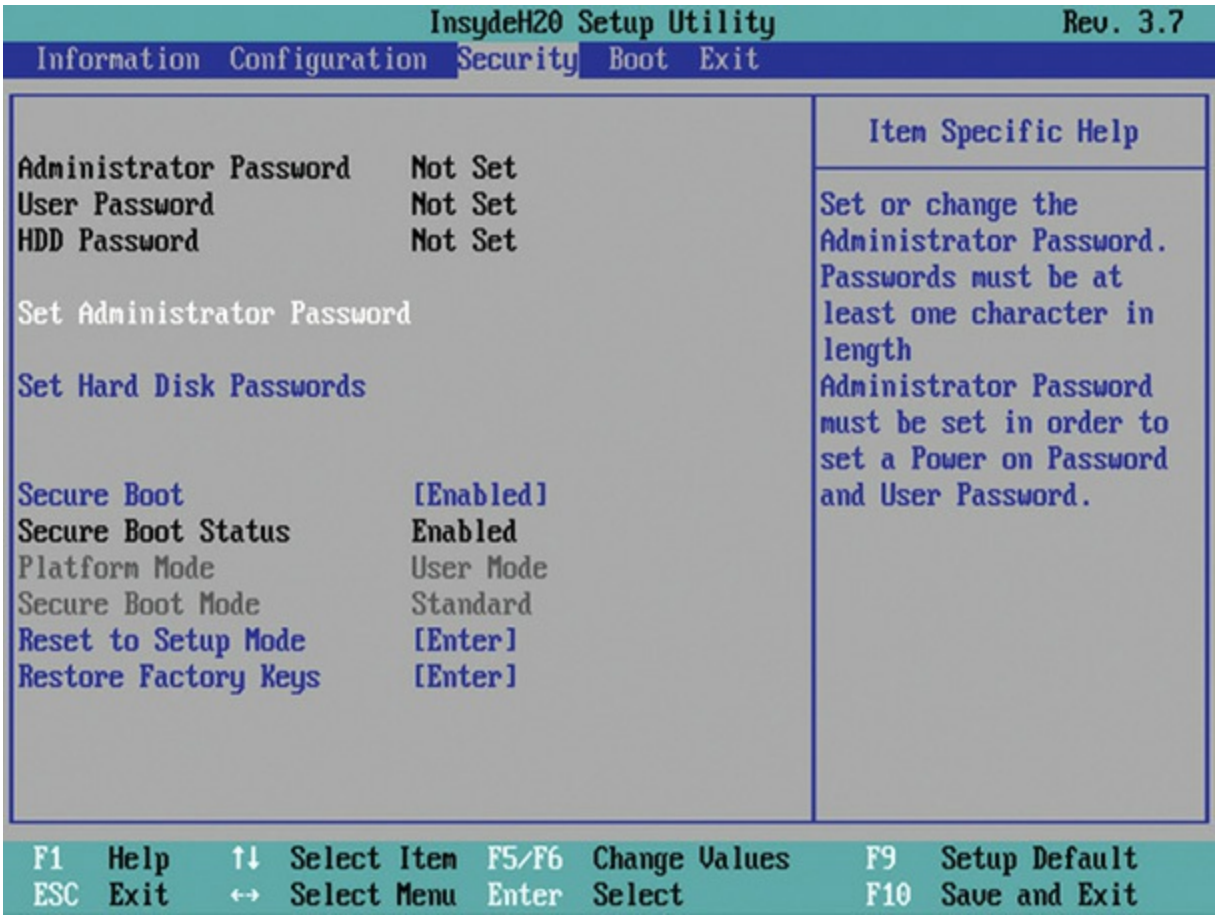

Figure 5-24 Security tab

The *Secure Boot* feature you can see on the Security tab is a UEFI protocol that secures the boot process by requiring properly signed software. This includes boot software and software that supports specific, essential components. (See “Device Drivers” a little later in this chapter.) Secure Boot requires an Intel CPU, a UEFI BIOS, and an operating system designed for it, such as Windows.

NOTE Secure Boot is an example of a tool that uses *drive encryption*. Various types of encryption—essentially scrambling the information to make it inaccessible to bad guys—secure all sorts of processes and data in modern

computing. We'll hit the subject in several places later in the book. Chapter 9, "Implementing Mass Storage," discusses drive encryption specifically in more detail.

The Boot tab (see Figure 5-25) enables you to set *boot options* to determine which bootable device gets priority. Here is where you provide support for booting to a USB device as well. It looks a little different from the graphical example presented earlier. See "The Boot Process" later in this chapter for more explanation.

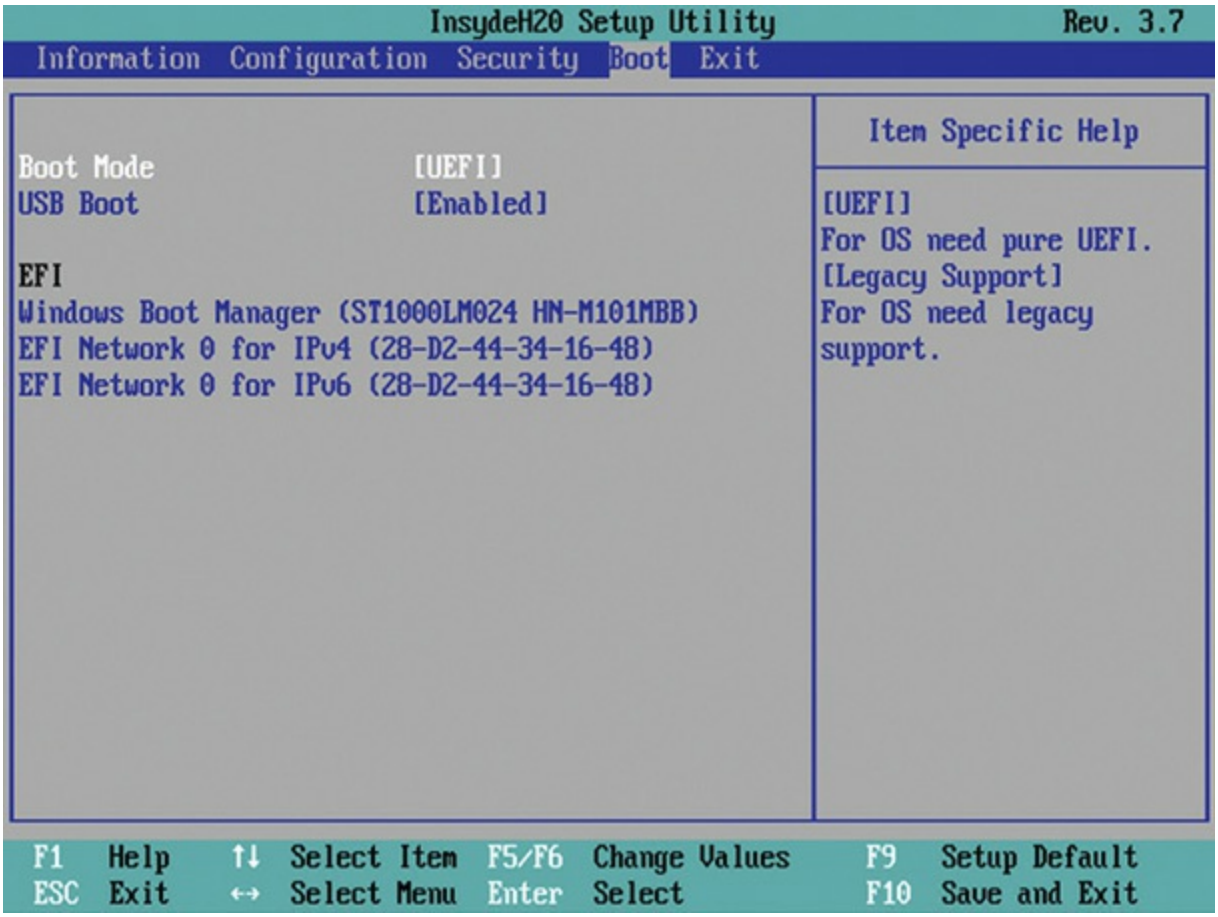

Figure 5-25 Boot tab

Other BIOS Security Settings

Motherboard manufacturers, BIOS/UEFI writers, and programmers have implemented all kinds of security features over the years. This section mentions a couple you might run into on various motherboards (or on a

certain exam in your near future).

Chassis Intrusion Detection/Notification

Many motherboards support the *chassis intrusion detection/notification* feature provided by the computer case, or chassis. Compatible cases contain a switch that trips when someone opens the case. With motherboard support and a proper connection between the motherboard and the case, the CMOS logs whether the case has been opened and, if it has, posts a notification to the screen on the subsequent boot. How cool is that?

LoJack

Some PC manufacturers include *LoJack* security features in their firmware—this way, if your PC is stolen, you can track its location, install a keylogger, or even remotely shut down your computer.

Trusted Platform Module

The *Trusted Platform Module* (*TPM*) acts as a secure cryptoprocessor, which is to say that it is a hardware platform for the acceleration of cryptographic functions and the secure storage of associated information. The specification for the TPM is published by the Trusted Computing Group, an organization whose corporate members include Intel, Microsoft, AMD, IBM, Lenovo, Dell, Hewlett-Packard, and many others.

The TPM can be a small circuit board plugged into the motherboard, or it can be built directly into the chipset. The CMOS setup program usually contains settings that can turn the TPM on or off and enable or disable it.

TPMs can be used in a wide array of cryptographic operations, but one of the most common uses of TPMs is hard disk encryption. For example, the *BitLocker Drive Encryption* feature of Microsoft Windows can be accelerated by a TPM, which is more secure because the encryption key is stored in the tamper-resistant TPM hardware rather than on an external flash drive. Other possible uses of TPMs include digital rights management (DRM), network access control, application execution control, and password protection.

EXAM TIP BIOS security-related options can include TPM, passwords, Secure Boot, intrusion detection/notification, and drive encryption.

Exiting and Saving Settings

Of course, all system setup utilities provide some method to Save, Exit Saving Changes, or Exit Discarding Changes (see Figure 5-26). Use these as needed for your situation. Exit Discarding Changes is particularly nice for those folks who want to poke around the CMOS setup utility but don't want to mess anything up. Use it!

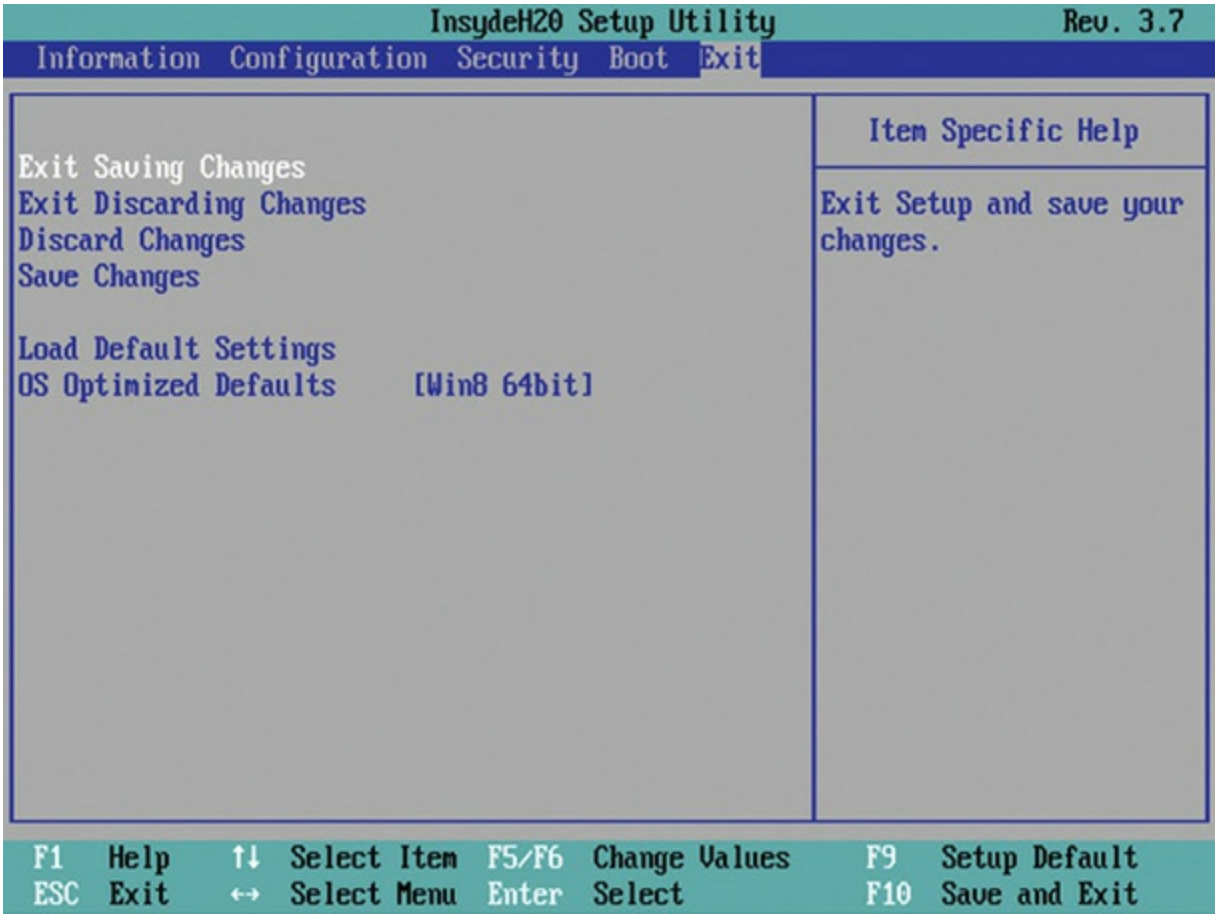

Figure 5-26 Exit options

The CMOS setup utility would meet all the needs of a modern system for BIOS if manufacturers would just stop creating new devices. That's not going to happen, of course, so let's turn now to devices that need to have BIOS loaded from elsewhere.

NOTE People serious about tweaking UEFI settings for maximum performance (overclocking) or minimum energy use (underclocking) can use a feature in some system setup utilities to save customized settings. Various utilities call them presets or profiles—essentially it's a "save these settings as" option. If something isn't quite right with the changes, go back into setup, make some changes, and try again. If you're done fiddling for the day and want to play with a stable machine, pick the profile you created that is the stable machine. See "Care and Feeding of BIOS/UEFI and CMOS" later in this chapter for the ultimate undo features.

Option ROM and Device Drivers

Every piece of hardware in your computer needs some kind of programming that tells the CPU how to talk to that device. When IBM invented the PC decades ago, they couldn't possibly have included all of the necessary BIOS routines for every conceivable piece of hardware on the system ROM chip. How could they? Most of the devices in use today didn't exist on the first PCs. When programmers wrote the first BIOS, for example, network cards, mice, and sound cards did not exist. Early PC designers at IBM understood that they could not anticipate every new type of hardware, so they gave us a few ways to add programming other than on the BIOS. I call this *BYOB*—Bring Your Own BIOS. You can BYOB in two ways: option ROM and device drivers. Let's look at both.

Option ROM

The first way to BYOB is to put the BIOS on the hardware device itself. Look at the card displayed in Figure 5-27. This is a serial ATA RAID hard

drive controller—basically just a card that lets you add more hard drives to a PC. The chip in the center with the wires coming out the sides is a flash ROM that stores BIOS for the card. The system BIOS does not have a clue about how to talk to this card, but that's okay, because this card brings its own BIOS on what's called an *option ROM* chip.

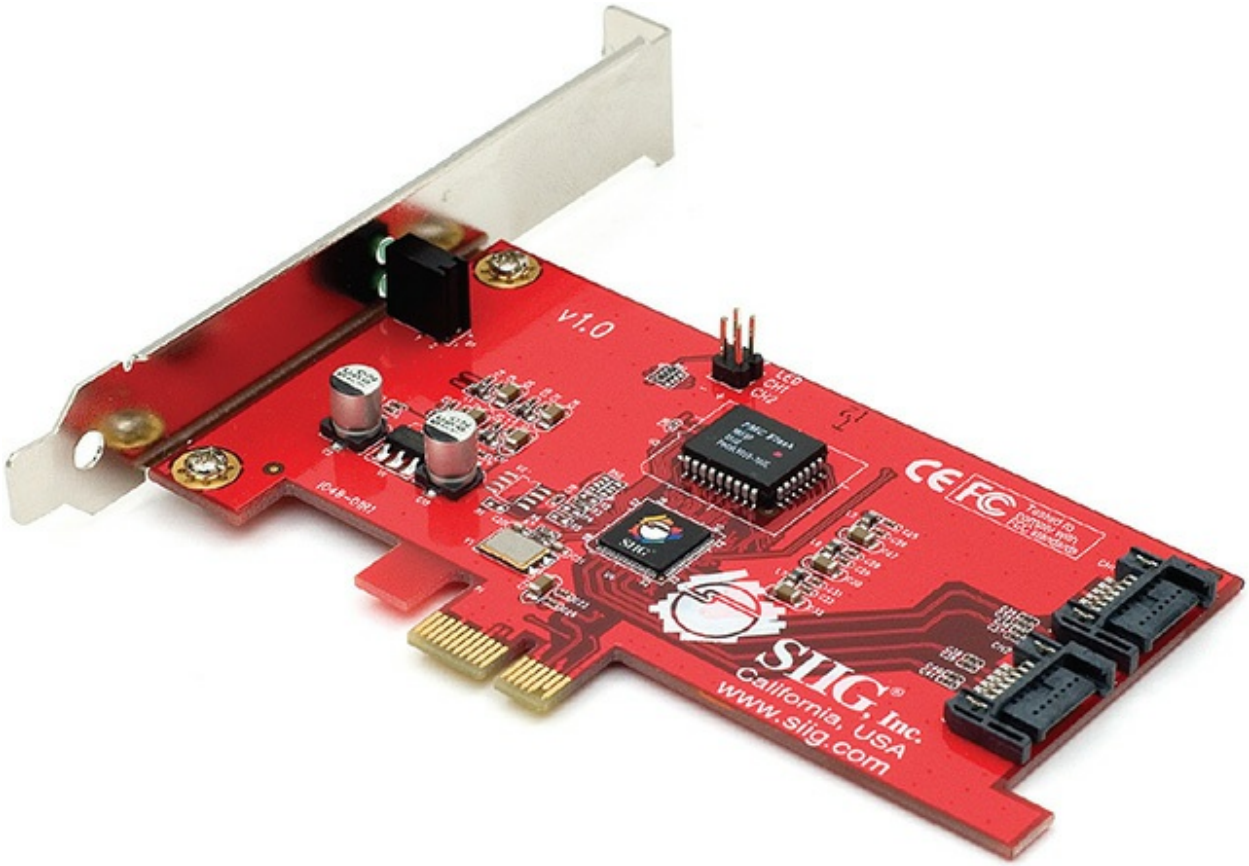

Figure 5-27 Option ROM

Most BIOS that come on option ROMs tell you that they exist by displaying information when you boot the system. Figure 5-28 shows a typical example of an option ROM advertising itself.

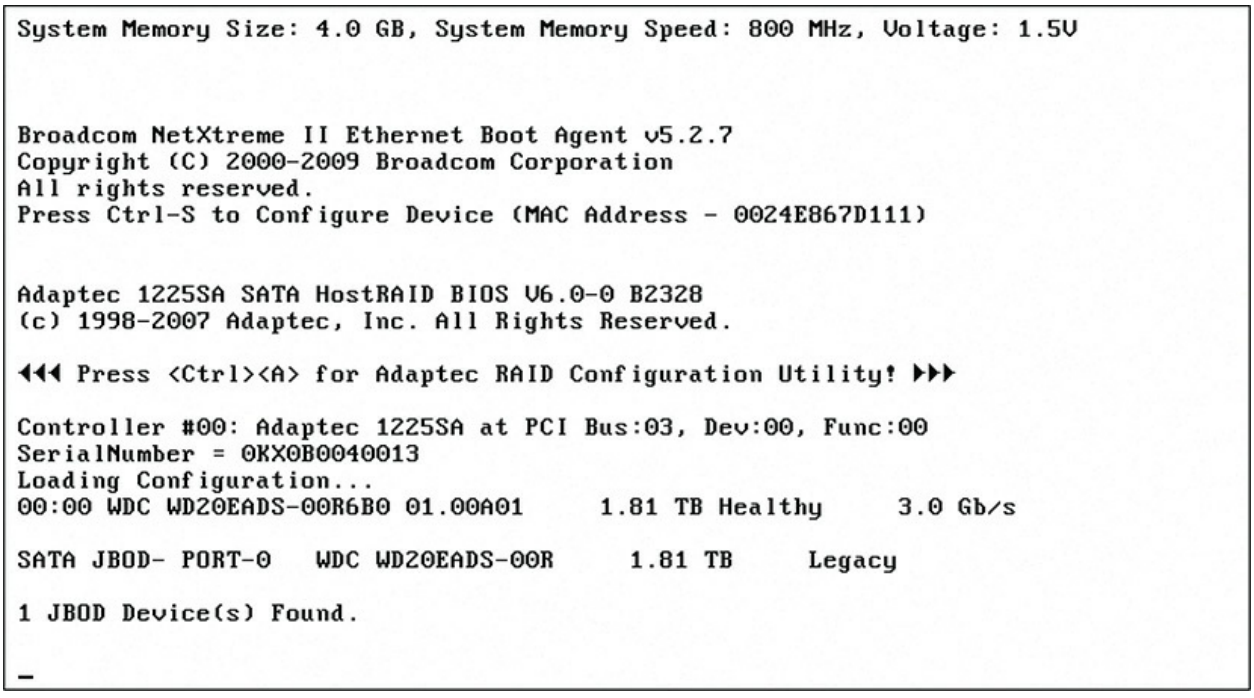

Figure 5-28 Option ROM at boot

In the early days of the PC, you could find all sorts of devices with BIOS on option ROMs. Today, option ROMs have mostly been replaced by more flexible software methods (more on device driver software in the next section), with one major exception: video cards. Every video card made today contains its own BIOS. Option ROMs work well but are hard to upgrade. For this reason, most hardware relies on software for BYOB.

Device Drivers

A *device driver* is a file stored on the PC's hard drive that contains all of the commands necessary to talk to whatever device it was written to support. All operating systems employ a method of loading these device drivers into RAM every time the system boots. They know which device drivers to install by reading a file (or files) that lists which device drivers the system needs to load at boot time. All operating systems are designed to look at this list early on in the boot process and copy the listed files into RAM, thereby giving the CPU (and the OS) the capability to communicate with the hardware supported by the device driver.

Device drivers come with the device when you buy it. When you buy

almost any device for your computer, that new sound card or monitor or whatever usually comes with some kind of media, often an optical disc, that holds all of the necessary device drivers (and usually a bunch of extra goodies—see Figure 5-29).

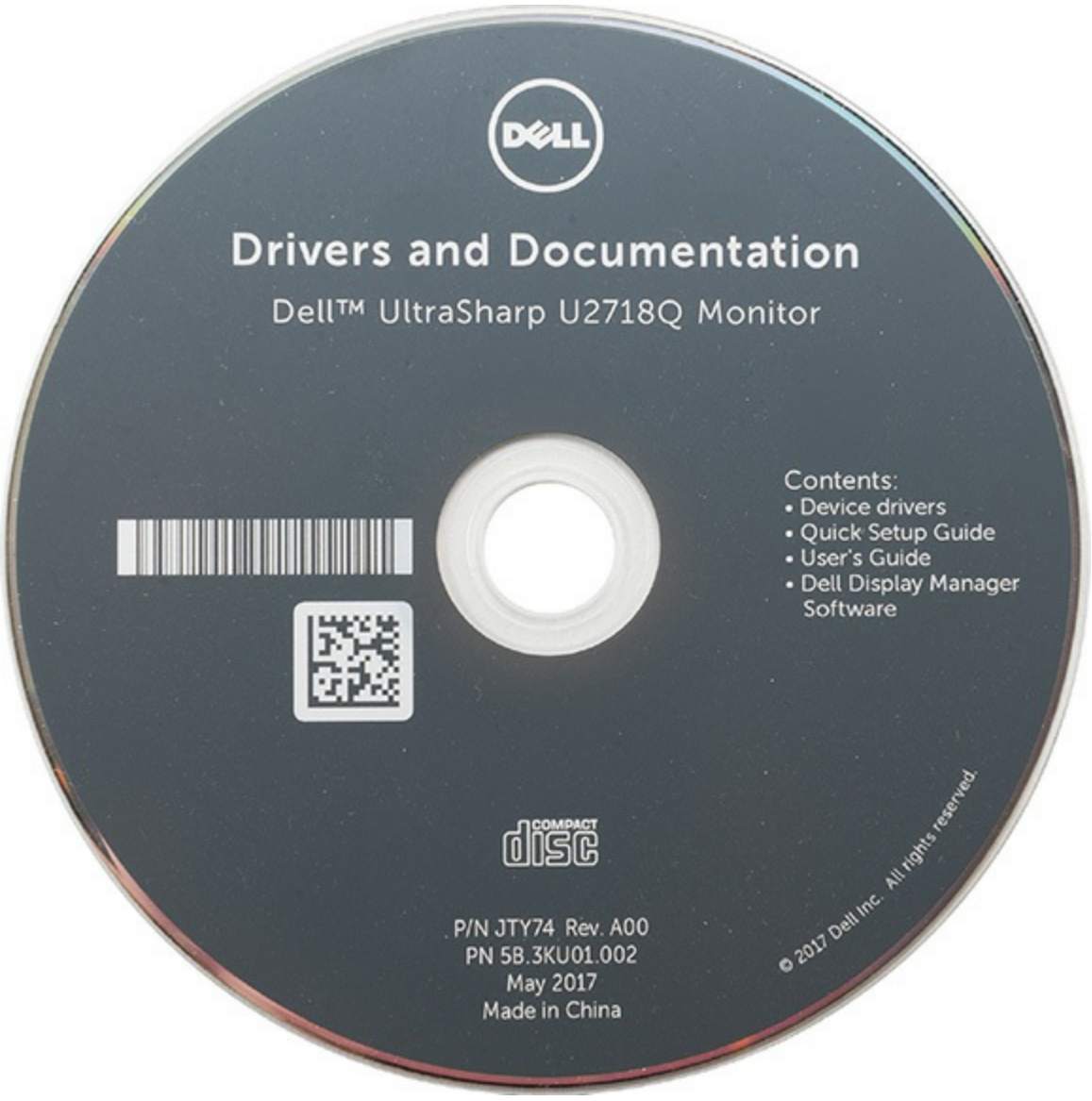

Figure 5-29 Installation disc for monitor

In many cases you may not want to use installation media and just let the OS handle things. All operating systems use online tools to detect and automatically install device drivers. You might want to add or remove device drivers manually at times. Windows stores device drivers (and a lot more) in

a database called the Registry, which we'll talk about in detail in Chapter 12, "Windows Under the Hood." Techs rarely deal directly with the Registry, but rather indirectly through the Device Manager utility (mentioned in Chapter 2, "The Visible Computer"). You'll see plenty more about Device Manager throughout this book.

BIOS, BIOS, Everywhere!

As you should now understand, every piece of hardware on a system must have an accompanying program that provides the CPU with the code necessary to communicate with that particular device. This code may reside on the system ROM on the motherboard, on ROM on a card, or in a device driver file on the hard drive loaded into RAM at boot. BIOS is everywhere on your system, and you need to deal with it occasionally.

Power-On Self Test (POST)

BIOS isn't the only program on system ROM. When the computer is turned on or reset, it initiates a special program, also stored on the system ROM chip, called the *power-on self test* (*POST*). The POST program checks out the system every time the computer boots. To perform this check, the POST sends out a command that says to all of the devices, "Check yourselves out!" All of the standard devices in the computer then run their own built-in diagnostic—the POST doesn't specify what they must check. The quality of the diagnostic is up to the people who made that particular device.

Let's consider the POST for a moment. Suppose some device—let's say it's the keyboard controller chip—runs its diagnostic and determines that it is not working properly. What can the POST do about it? Only one thing really: tell the human in front of the PC! So how does the computer tell the human? PCs convey POST information to you in two ways: beep codes and text messages.

Before and During the Video Test: The Beep Codes

The computer tests the most basic parts of the computer first, up to and including the video card. In early PCs, you'd hear a series of beeps—called *beep codes* or *POST beep codes*—if anything went wrong. By using beep

codes before and during the video test, the computer could communicate with you. (If a POST error occurs before the video is available, obviously the error must manifest itself as beeps, because nothing can display on the screen.) The meaning of the beep code you'd hear varied among different BIOS manufacturers. You could find the beep codes for a specific motherboard in its motherboard manual.

NOTE CompTIA refers to beep codes as *POST code beeps*.

Most modern PCs have only two beep codes: one for bad or missing video (one long beep followed by two or three short beeps), and one for bad or missing RAM (a single beep that repeats indefinitely).

CAUTION You'll find lots of online documentation about beep codes, but it's usually badly outdated.

You'll hear three other beep sequences on most PCs (although they're not officially beep codes). At the end of a successful POST, the PC produces one or two short beeps, simply to inform you that all is well. Most systems make a rather strange noise when the RAM is missing or very seriously damaged. Unlike traditional beep codes, this code repeats until you shut off the system. Finally, your speaker might make beeps for reasons that aren't POST or boot related. One of the more common is a series of short beeps after the system's been running for a while. That's a CPU alarm telling you the CPU is approaching its high heat limit.

Text Errors

After the video has tested okay, any POST errors display on the screen as text

errors. If you get a text error, the problem is usually, but not always, self-explanatory (see Figure 5-30). Text errors are far more useful than beep codes, because you can simply read the screen to determine the bad device.

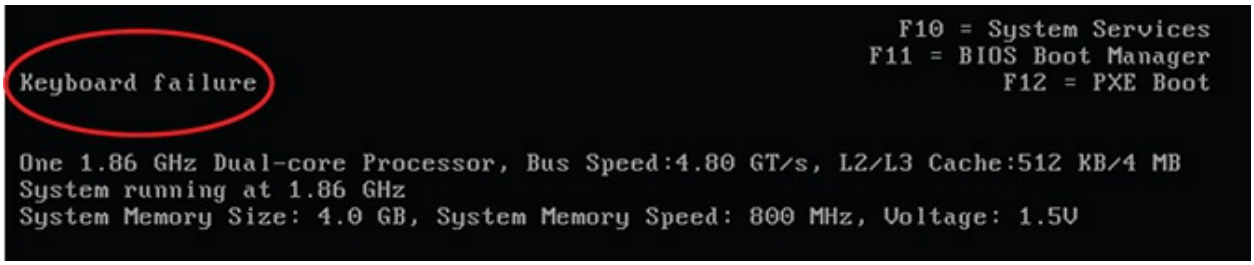

Figure 5-30 POST text error messages

POST Cards

Beep codes, numeric codes, and text error codes, although helpful, can sometimes be misleading. Worse than that, an inoperative device can sometimes disrupt the POST, forcing the machine into an endless loop. This causes the PC to act dead—no beeps and nothing on the screen. In this case, you need a device, called a *POST card*, to monitor the POST and identify which piece of hardware is causing the trouble.

POST cards are simple cards that snap into expansion slots on your system. A small, two-character light-emitting diode (LED) readout on the card indicates which device the POST is currently testing (see Figure 5-31).

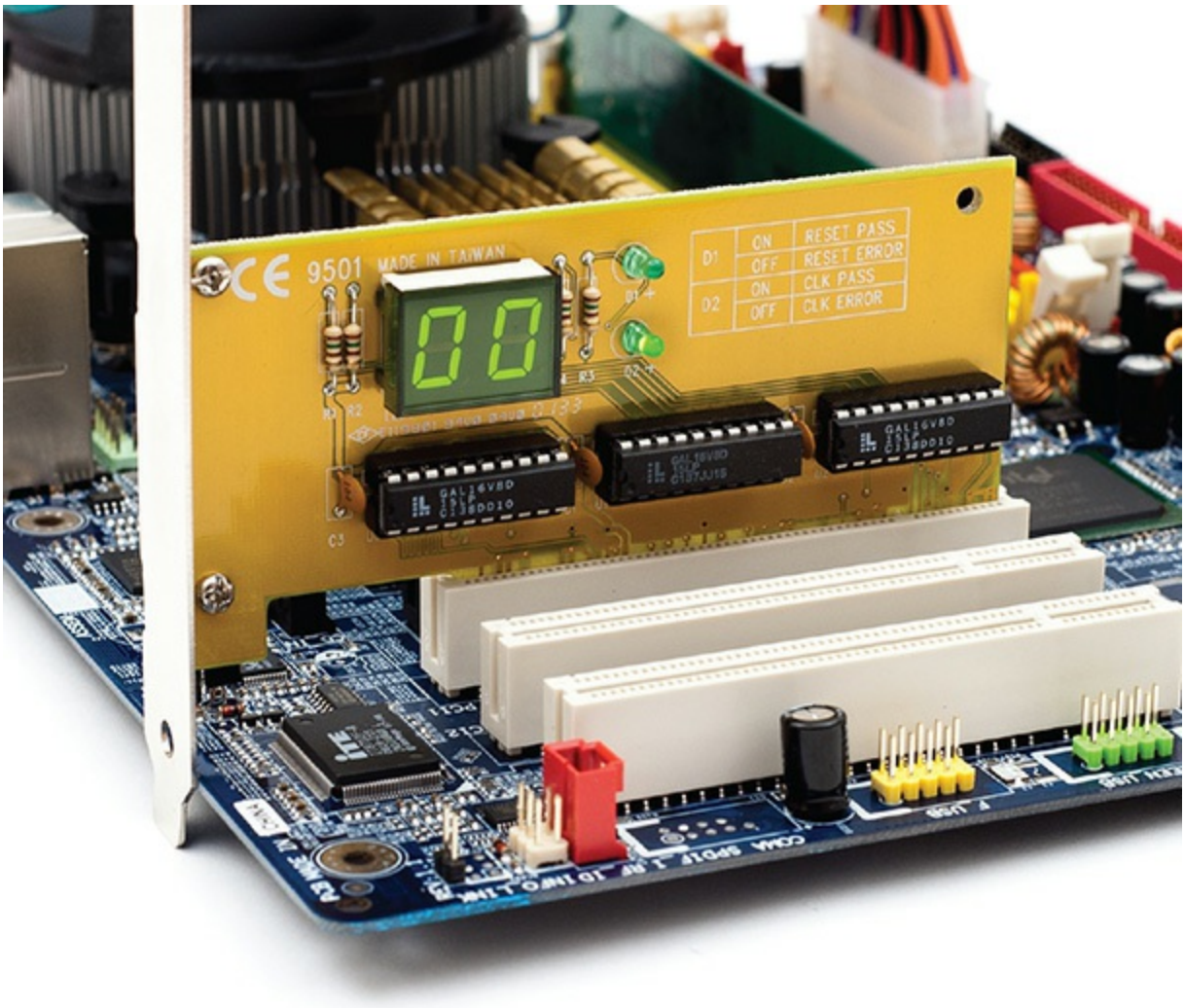

Figure 5-31 POST card in action

POST cards used to be essential tools for techs, but today I use them only when I have a “dead” PC to determine at which level it’s dead. If the POST card shows no reading, I know the problem is before the POST and must be related to the power, the CPU, the RAM, or the motherboard. If the board posts, then I know to look at more issues, such as the drives and so on.

The Boot Process

All PCs need a process to begin their operations. Once you feed power to the PC, the tight interrelation of hardware, firmware, and software enables the PC to start itself, to “pull itself up by the bootstraps” or boot itself.

When you first power on the PC, the power supply circuitry tests for proper voltage and then sends a signal down a special wire called the *power good* wire to awaken the CPU. The moment the power good wire wakes it up, every Intel and clone CPU immediately sends a built-in memory address via its address bus. This special address is the same on every Intel and clone CPU, from the oldest 8086 to the most recent microprocessor. This address is the first line of the POST program on the system ROM! That's how the system starts the POST. After the POST has finished, there must be a way for the computer to find the programs on the hard drive to start the operating system. What happens next differs between the old BIOS way and the UEFI way.

In the older BIOS environment, the POST passes control to the last BIOS function: the bootstrap loader. The *bootstrap loader* is little more than a few dozen lines of BIOS code tacked to the end of the POST program. Its job is to find the operating system. The bootstrap loader reads CMOS information to tell it where to look first for an operating system. Your PC's CMOS setup utility has an option that you configure to tell the bootstrap loader which devices to check for an operating system and in which order—that's the *boot sequence* (see Figure 5-32).

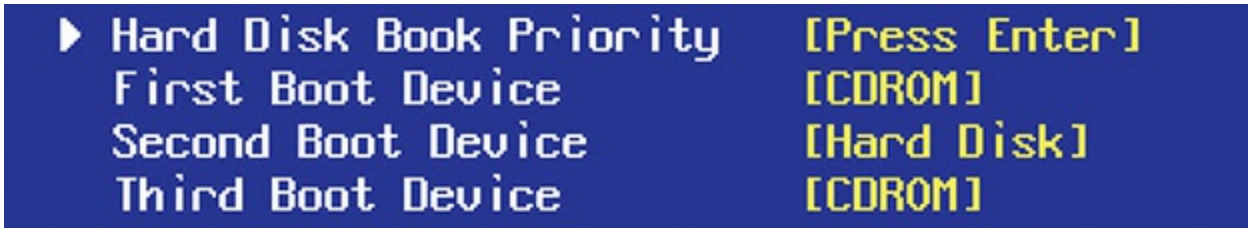

Figure 5-32 CMOS boot sequence

Almost all storage devices—hard disk drives, solid-state drives, CDs, DVDs, and USB thumb drives—can be configured to boot an operating system by setting aside a specific location called the *boot sector*. If the device is bootable, its boot sector contains special programming designed to tell the system where to locate the operating system. Any device with a functional operating system is called a *bootable disk* or a *system disk*. If the bootstrap loader locates a good boot sector, it passes control to the operating system and removes itself from memory. If it doesn't, it goes to the next device in the boot sequence you set in the CMOS setup utility. The boot sequence is an important tool for techs because you can set it to load in special bootable

devices so you can run utilities to maintain PCs without using the primary operating system.

In UEFI systems, the POST hands control of the boot process to the Boot Manager, which checks the boot configuration, and then loads the operating system boot loader directly (see Figure 5-33). There's no need for scanning for a boot sector or any of that. UEFI firmware stores the boot manager and boot configuration.

Figure 5-33 UEFI Boot Mode with Boot Manager options displayed

NOTE If you put an old-style BIOS bootable disk in a UEFI system, the system will most likely drop into BIOS compatibility mode and boot just like the old days. See Chapter 9 for more on drive structures.

Some BIOS include a feature that enables a PC to use a *preboot execution environment* (*PXE*). A PXE enables you to boot a PC without any local storage by retrieving an OS from a server over a network. You'll see more on

PXE when we talk about installing Windows in Chapter 11, “Building a PC.”

Care and Feeding of BIOS/UEFI and CMOS

BIOS and CMOS are areas in your PC that you don’t go to very often. BIOS itself is invisible. The only real clue you have that it even exists is the POST. The CMOS setup utility, on the other hand, is very visible if you start it. Most CMOS setup utilities today work acceptably well without ever being touched. You’re an aspiring tech, however, and all self-respecting techs start up the CMOS setup utility and make changes. That’s when most CMOS setup utility problems take place.

If you mess with the CMOS setup utility, remember to make only as many changes at one time as you can remember. Document the original settings and the changes on a piece of paper or take a photo so you can put things back if necessary. Don’t make changes unless you know what they mean! It’s easy to screw up a computer fairly seriously by playing with CMOS settings you don’t understand.

Default/Optimized Settings

Every CMOS setup utility has a couple of reset options, commonly called Load Default Settings and OS Optimized Defaults (see Figure 5-34). These options keep you from having to memorize all of those weird settings you’ll never touch. Default or Fail-Safe sets everything to very simple settings—you might occasionally use this setting when very low-level problems such as freeze-ups occur and you’ve checked more obvious areas first. Optimized sets the CMOS to the best possible speed/stability for the system. You would use this option after you’ve tampered with the CMOS too much and need to put it back like it was!

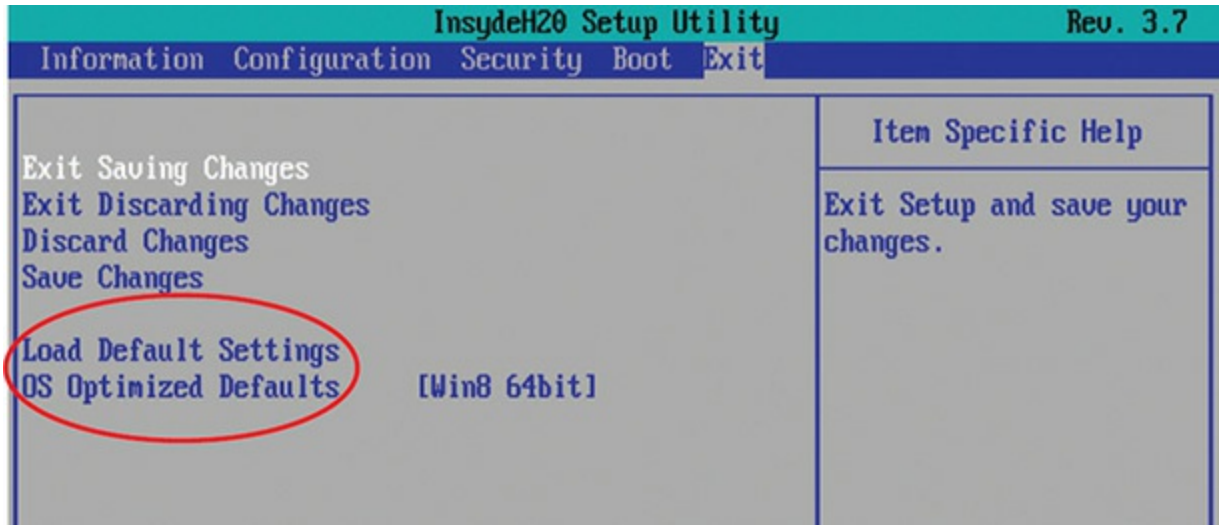

Figure 5-34 Options for resetting CMOS

Clearing CMOS RTC RAM

You read about the process for clearing system settings back in Chapter 3, but the process is worth repeating here. When you mess up a setting (by overclocking too much or disabling something that should have remained enabled—or vice versa) that renders the computer dead, you can reset the CMOS RTC RAM back to factory defaults and start over.

Almost every motherboard has a dedicated set of wires called *CLRTC* or something similar (see Figure 5-35).

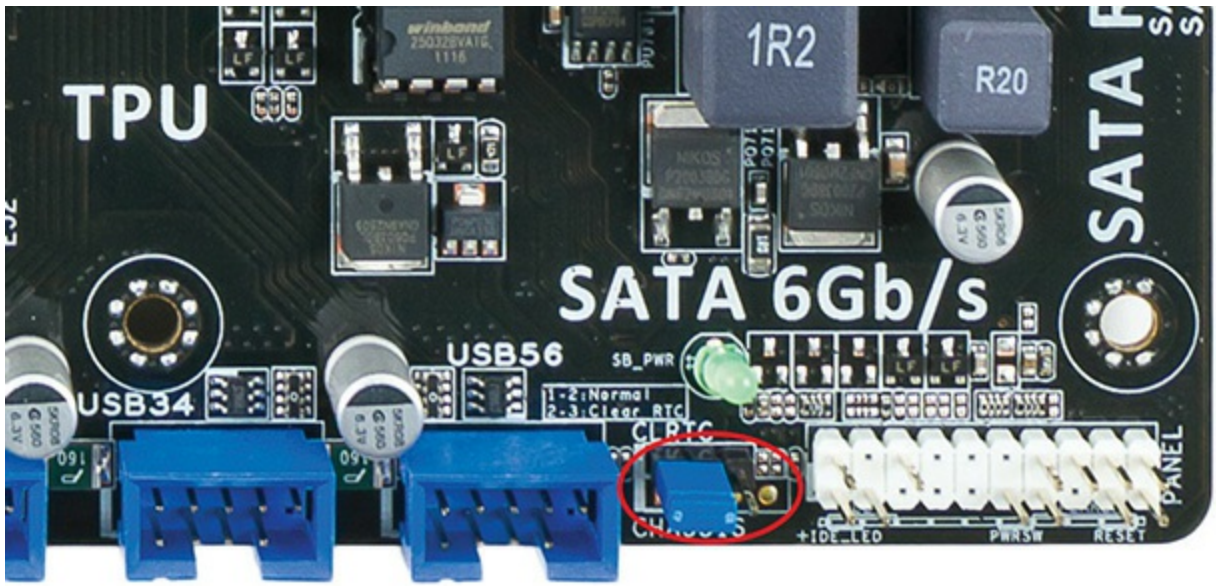

Figure 5-35 CMOS RTC clear wires

NOTE Many techs use older language to describe the reset CMOS RTC RAM, simply *CMOS clear*, describing both the process and the motherboard option.

Turn off and unplug the computer, then open the case to access the motherboard. Find the CMOS RTC clear wires. Move the shunt (the little plastic and metal jumper thing) from wires 1 and 2 to wires 2 and 3 (see Figure 5-36). Wait for 10 seconds and then move the shunt back to the default position. Plug in and boot the system.

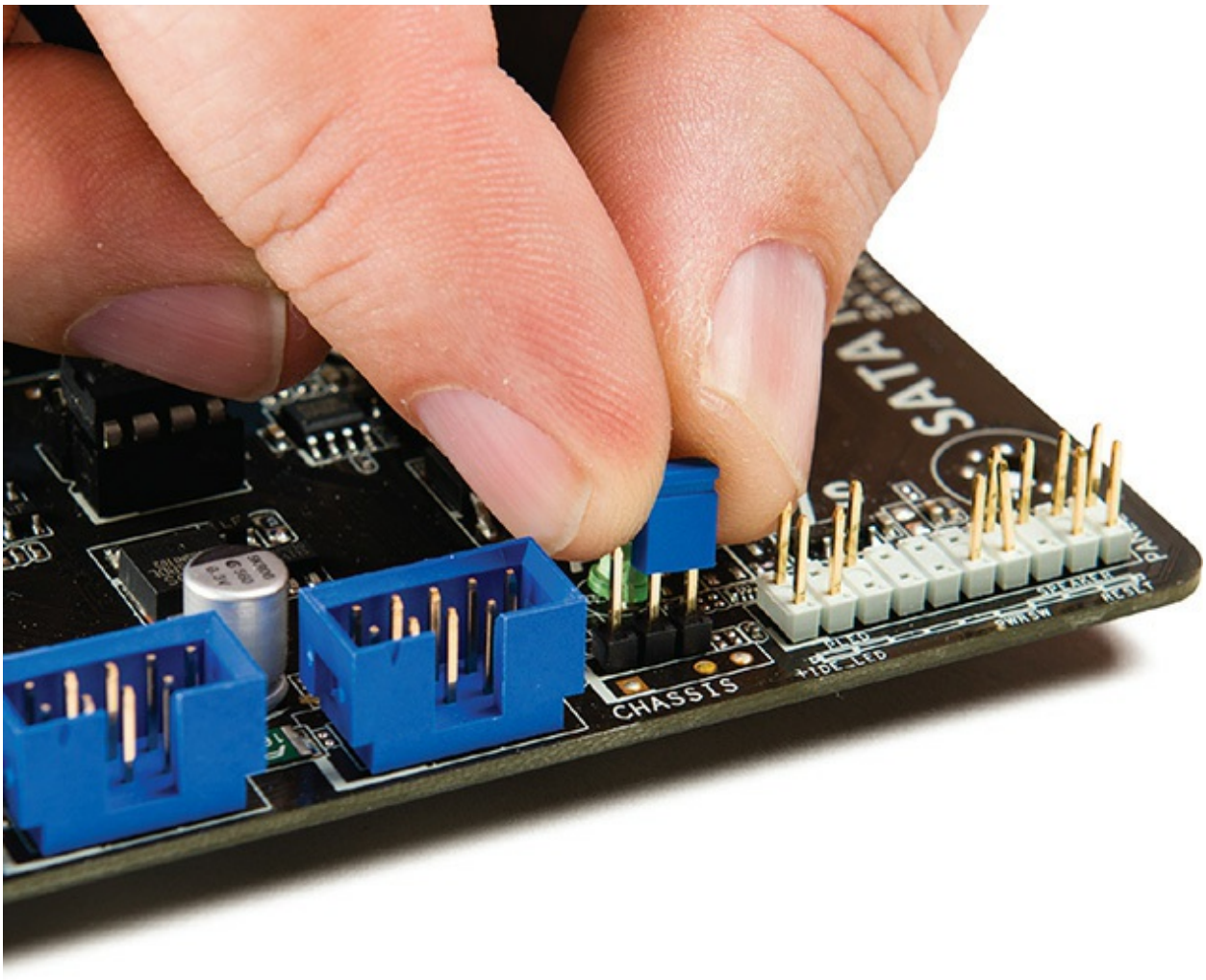

Figure 5-36 Changing shunt location to clear CMOS RAM

NOTE Manufacturers of enthusiast boards designed for easy overclocking experimentation know you're going to screw up during the overclocking process. You'll often find a dedicated clear CMOS button hardwired to the motherboard. Now that's service!

If that doesn't work or if you get one of the truly odd motherboards without CLRTC jumpers, power down the system and unplug. Pry out the little coin battery (see below) and wait for several seconds. Reinstall and

reboot.

Losing CMOS RTC Settings

As mentioned before, your CMOS RAM needs a continuous trickle charge to keep the internal clock running and remember its settings. Motherboards use some type of battery, usually a 3-volt Lithium coin battery, to give the CMOS RAM the charge it needs when the computer is turned off (see Figure 5-37). This is called the *CMOS battery*. Typical systems use a *CR2032* battery. (What does your system use?)

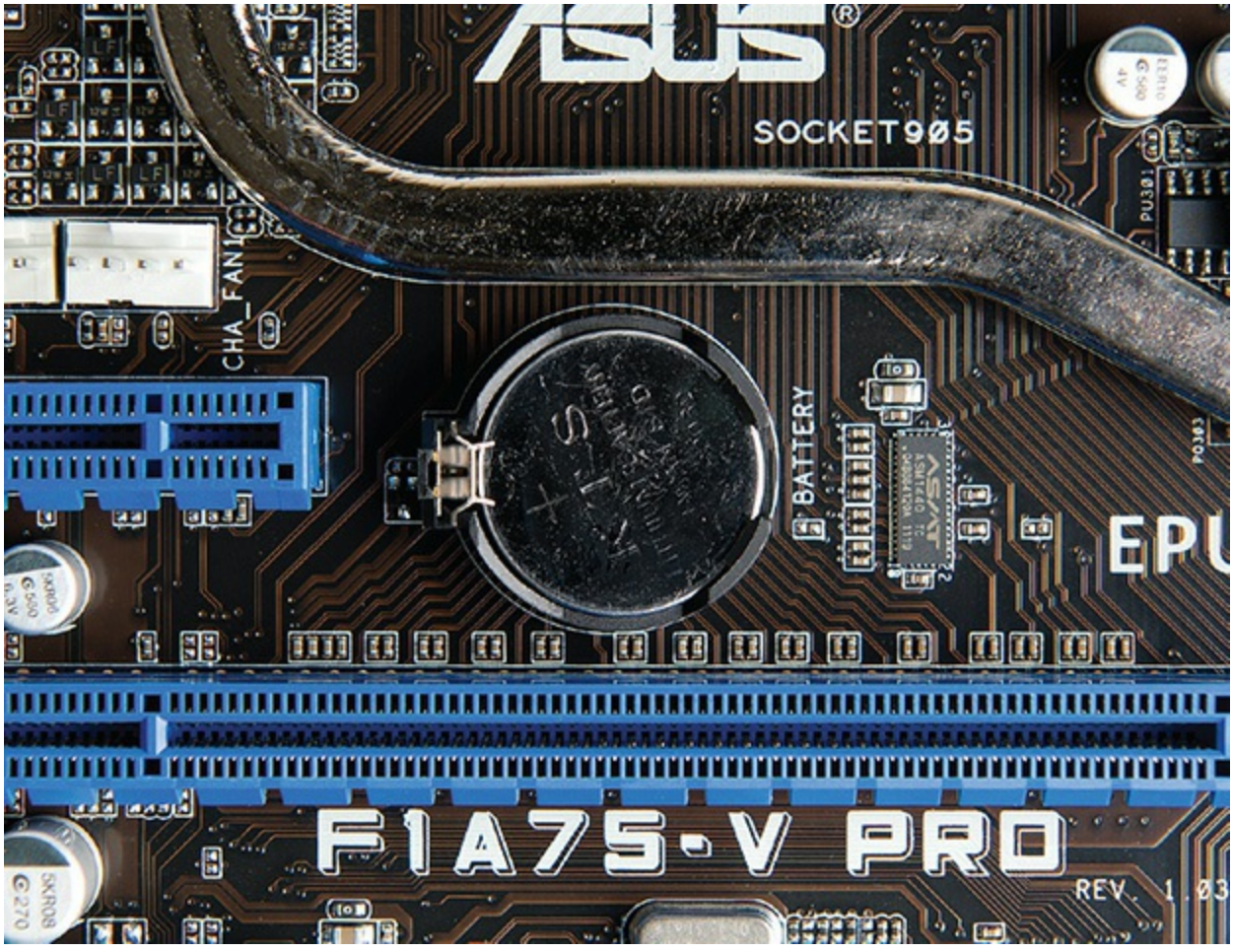

Figure 5-37 A CMOS battery

If some mishap suddenly erases the information on the CMOS RAM, the computer might not boot or you'll get nasty-looking errors at boot. Any PC will boot to factory defaults if the CMOS clears, so the chances of not

booting are slim—but you'll still get errors at boot. Here are a few examples of errors that point to a lost CMOS information scenario:

- CMOS configuration mismatch
- CMOS date/time not set
- BIOS time and settings reset
- No boot device available
- CMOS battery state low

Here are some of the more common reasons for losing CMOS data:

- Pulling and inserting cards
- Touching the motherboard
- Dropping something on the motherboard
- Dirt on the motherboard
- Faulty power supplies
- Electrical surges

If you run into any of these scenarios, or if the clock in Windows resets itself to January 1st every time you reboot the system, the battery on the motherboard is losing its charge and needs to be replaced. To replace the battery, use a screwdriver to pry the battery's catch gently back. The battery should pop up for easy removal. Before you install the new battery, double-check that it has the same voltage and amperage as the old battery. To retain your CMOS settings while replacing the battery, simply leave your PC plugged into an AC outlet. The 5-volt soft power on all modern motherboards provides enough electricity to keep the CMOS charged and the data secure. Of course, I know you're going to be *extremely* careful about ESD while prying up the battery from a live system!

Flashing the ROM

Flash ROM chips can be reprogrammed to update their contents. With flash ROM, when you need to update your system BIOS to add support for a new technology, you can simply run a small command-line program, combined with an update file, and voilà, you have a new, updated BIOS! This is called

a *firmware update*. Different BIOS makers use slightly different processes for *flashing the BIOS*, but, in general, you insert a removable disk of some sort (usually a USB thumb drive) containing an updated BIOS file and use the updating utility in CMOS setup.

Some motherboard makers provide Windows-based flash ROM update utilities that check the Internet for updates and download them for you to install. Most of these utilities also enable you to back up your current BIOS so you can return to it if the updated version causes trouble. Without a good backup, you could end up throwing away your motherboard if a flash BIOS update goes wrong, so you should always make one.

Finally, a lot of motherboards these days have system setup utilities that can connect directly to the Internet and access updates that way. Figure 5-38 shows one such update utility.

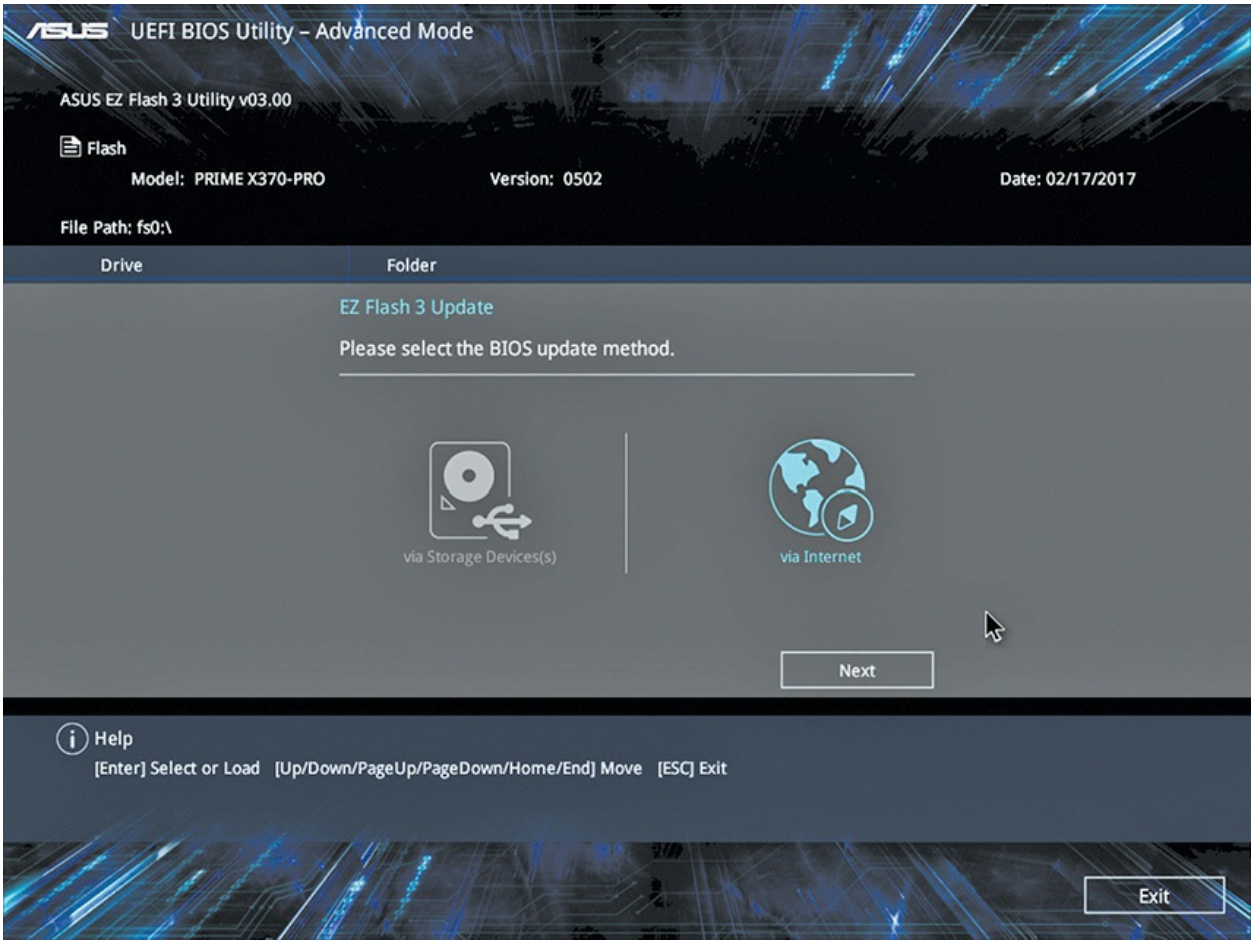

Figure 5-38 ROM-updating program for an ASUS motherboard

NOTE A failed BIOS/UEFI update—where something goes wrong during the process—can *brick* a computer or device. The failure turns a computing device into a brick, useless for anything but a paperweight.

Just a word of caution to complete the BIOS update section. Don't update your BIOS unless you have some compelling reason to do so. As the old saying goes, "If it ain't broke, don't fix it!"

EXAM TIP While techs usually talk about "flashing the BIOS," the CompTIA A+ exams refer to this process also as "firmware updates."

Chapter Review

Questions

1. What does BIOS provide for the computer? (Choose the best answer.)
   - **A.** BIOS provides the physical interface for various devices such as USB and FireWire ports.
   - **B.** BIOS provides the programming that enables the CPU to communicate with other hardware.
   - **C.** BIOS provides memory space for applications to load into from the hard drive.
   - **D.** BIOS provides memory space for applications to load into from the main system RAM.
2. What is the correct boot sequence for an older BIOS-based PC?
   - **A.** CPU, POST, power good, boot loader, operating system

B. POST, power good, CPU, boot loader, operating system

C. Power good, boot loader, CPU, POST, operating system

D. Power good, CPU, POST, boot loader, operating system

3. Jill decided to add a second hard drive to her computer. She thinks she has it physically installed correctly, but it doesn't show up in Windows. Which of the following options will most likely lead Jill where she needs to go to resolve the issue?

A. Reboot the computer and press the F key on the keyboard twice. This signals that the computer has two hard disk drives.

B. Reboot the computer and watch for instructions to enter the CMOS setup utility (for example, a message may say to press the DELETE key). Do what it says to go into CMOS setup.

B. In Windows, press the DELETE key twice to enter the CMOS setup utility.

B. In Windows, go to Start | Run and type **hard drive**. Click OK to open the Hard Drive Setup Wizard.

4. Henry bought a new card for capturing television on his computer. When he finished going through the packaging, though, he found no driver disc, only an application disc for setting up the TV capture software. After installing the card and software, it all works flawlessly. What's the most likely explanation?

A. The device doesn't need BIOS, so there's no need for a driver disc.

B. The device has an option ROM that loads BIOS, so there's no need for a driver disc.

C. Windows supports TV capture cards out of the box, so there's no need for a driver disc.

D. The manufacturer made a mistake and didn't include everything needed to set up the device.

5. Which of the following most accurately describes the relationship between BIOS and hardware?

A. All hardware needs BIOS.

B. All hardware that attaches to the motherboard via ribbon cables

needs BIOS.

C. All hardware built into the motherboard needs BIOS.

D. Some hardware devices need BIOS.

6. After a sudden power outage, Samson's PC rebooted, but nothing appeared on the screen. The PC just beeps at him, over and over and over. What's most likely the problem?

A. The power outage toasted his RAM.

B. The power outage toasted his video card.

C. The power outage toasted his hard drive.

D. The power outage toasted his CPU.

7. Davos finds that a disgruntled former employee decided to sabotage her computer when she left by putting a password in CMOS that stops the computer from booting. What can Davos do to solve this problem?

A. Davos should boot the computer while holding the left SHIFT key. This will clear the CMOS information.

B. Davos should try various combinations of the former employee's name. The vast majority of people use their name or initials for CMOS passwords.

C. Davos should find the CLRTC jumper on the motherboard. Then he can boot the computer with a shunt on the jumper to clear the CMOS information.

D. Davos should find a replacement motherboard. Unless he knows the CMOS password, there's nothing he can do.

8. Richard over in the sales department went wild in CMOS and made a bunch of changes that he thought would optimize his PC. Now most of his PC doesn't work. The computer powers up, but he can only get to CMOS, not into Windows. Which of the following tech call answers would most likely get him up and running again?

A. Reboot the computer about three times. That'll clear the CMOS and get you up and running.

B. Open up the computer and find the CLRTC jumper. Remove a shunt from somewhere on the motherboard and put it on the

CLRTC jumper. Reboot and then put the shunt back where you got it. Reboot, and you should be up and running in no time.

C. Boot into the CMOS setup program and then find the option to load a plug-and-play operating system. Make sure it's set to On. Save and exit CMOS; boot normally into Windows. You should be up and running in no time.

D. Boot into the CMOS setup program and then find the option to load OS Optimized Defaults. Save and exit CMOS; boot normally into Windows. You should be up and running in no time.

9. Jill boots an older Pentium system that has been the cause of several user complaints at the office. The system powers up and starts to run through POST, but then stops. The screen displays a "CMOS configuration mismatch" error. Of the following list, what is the most likely cause of this error?

A. Dying CMOS battery

B. Bad CPU

C. Bad RAM

D. Corrupt system BIOS

10. Where does Windows store device drivers?

A. Computer

B. Hardware

C. Registry

D. Drivers and Settings

Answers

1. **B.** BIOS provides the programming that enables the CPU to communicate with other hardware.

2. **D.** Here's the correct boot sequence for a BIOS-based PC: power good, CPU, POST, boot loader, operating system.

3. **B.** Jill should reboot the computer and watch for instructions to enter the CMOS setup utility (for example, a message may say to press the DELETE key). She should do what it says to go into CMOS setup.

4. **B.** Most likely the device has an option ROM, because it works.
5. **A.** All hardware needs BIOS!
6. **A.** The long repeating beep and a dead PC most likely indicate a problem with RAM.
7. **C.** Davos should find the CLRTC jumper on the motherboard and then boot the computer with a shunt on the jumper to clear the CMOS information.
8. **D.** Please don't hand Richard a screwdriver! Having him load Optimized Default settings will most likely do the trick.
9. **A.** The CMOS battery is likely dying.
10. **C.** Windows stores device drivers in the Registry.