

"Computing is not about computers anymore. It is about living. . . . We have seen computers move out of giant air-conditioned rooms into closets, then onto desktops, and now into our laps and pockets. But this is not the end. . . . Like a force of nature, the digital age cannot be denied or stopped. . . . The information superhighway may be mostly hype today, but it is an understatement about tomorrow. It will exist beyond people's wildest predictions. . . . We are not waiting on any invention. It is here. It is now. It is almost genetic in its nature, in that each generation will become more digital than the preceding one."

—Nicholas Negroponte, professor of media technology at MIT

CHAPTER

1

Introduction

1.1 OVERVIEW

Dr. Negroponte is among many who see the computer revolution as if it were a force of nature. This force has the potential to carry humanity to its digital destiny, allowing us to conquer problems that have eluded us for centuries, as well as all of the problems that emerge as we solve the original problems. Computers have freed us from the tedium of routine tasks, liberating our collective creative potential so that we can, of course, build bigger and better computers.

As we observe the profound scientific and social changes that computers have brought us, it is easy to start feeling overwhelmed by the complexity of it all. This complexity, however, emanates from concepts that are fundamentally very simple. These simple ideas are the ones that have brought us to where we are today and are the foundation for the computers of the future. To what extent they will survive in the future is anybody's guess. But today, they are the foundation for all of computer science as we know it.

Computer scientists are usually more concerned with writing complex program algorithms than with designing computer hardware. Of course, if we want our algorithms to be useful, a computer eventually has to run them. Some algorithms are so complicated that they would take too long to run on today's systems. These kinds of algorithms are considered **computationally infeasible**. Certainly, at the current rate of innovation, some things that are infeasible today could be feasible tomorrow, but it seems that no matter how big or fast computers become, someone will think up a problem that will exceed the reasonable limits of the machine.

To understand why an algorithm is infeasible, or to understand why the implementation of a feasible algorithm is running too slowly, you must be able to see the program from the computer's point of view. You must understand what

makes a computer system tick before you can attempt to optimize the programs that it runs. Attempting to optimize a computer system without first understanding it is like attempting to tune your car by pouring an elixir into the gas tank: You'll be lucky if it runs at all when you're finished.

Program optimization and system tuning are perhaps the most important motivations for learning how computers work. There are, however, many other reasons. For example, if you want to write compilers, you must understand the hardware environment within which the compiler will function. The best compilers leverage particular hardware features (such as pipelining) for greater speed and efficiency.

If you ever need to model large, complex, real-world systems, you will need to know how floating-point arithmetic should work as well as how it really works in practice. If you wish to design peripheral equipment or the software that drives peripheral equipment, you must know every detail of how a particular computer deals with its input/output (I/O). If your work involves embedded systems, you need to know that these systems are usually resource-constrained. Your understanding of time, space, and price trade-offs, as well as I/O architectures, will be essential to your career.

All computer professionals should be familiar with the concepts of benchmarking and be able to interpret and present the results of benchmarking systems. People who perform research involving hardware systems, networks, or algorithms find benchmarking techniques crucial to their day-to-day work. Technical managers in charge of buying hardware also use benchmarks to help them buy the best system for a given amount of money, keeping in mind the ways in which performance benchmarks can be manipulated to imply results favorable to particular systems.

The preceding examples illustrate the idea that a fundamental relationship exists between computer hardware and many aspects of programming and software components in computer systems. Therefore, regardless of our areas of expertise, as computer scientists, it is imperative that we understand how hardware interacts with software. We must become familiar with how various circuits and components fit together to create working computer systems. We do this through the study of **computer organization**. Computer organization addresses issues such as control signals (how the computer is controlled), signaling methods, and memory types. It encompasses all physical aspects of computer systems. It helps us to answer the question: How does a computer work?

The study of **computer architecture**, on the other hand, focuses on the structure and behavior of the computer system and refers to the logical and abstract aspects of system implementation as seen by the programmer. Computer architecture includes many elements such as instruction sets and formats, operation codes, data types, the number and types of registers, addressing modes, main memory access methods, and various I/O mechanisms. The architecture of a system directly affects the logical execution of programs. Studying computer architecture helps us to answer the question: How do I design a computer?

The computer architecture for a given machine is the combination of its hardware components plus its **instruction set architecture (ISA)**. The ISA is the agreed-upon interface between all the software that runs on the machine and the hardware that executes it. The ISA allows you to talk to the machine.

The distinction between computer organization and computer architecture is not clear-cut. People in the fields of computer science and computer engineering hold differing opinions as to exactly which concepts pertain to computer organization and which pertain to computer architecture. In fact, neither computer organization nor computer architecture can stand alone. They are interrelated and interdependent. We can truly understand each of them only after we comprehend both of them. Our comprehension of computer organization and architecture ultimately leads to a deeper understanding of computers and computation—the heart and soul of computer science.

1.2 THE MAIN COMPONENTS OF A COMPUTER

Although it is difficult to distinguish between the ideas belonging to computer organization and those ideas belonging to computer architecture, it is impossible to say where hardware issues end and software issues begin. Computer scientists design algorithms that usually are implemented as programs written in some computer language, such as Java or C++. But what makes the algorithm run? Another algorithm, of course! And another algorithm runs that algorithm, and so on until you get down to the machine level, which can be thought of as an algorithm implemented as an electronic device. Thus, modern computers are actually implementations of algorithms that execute other algorithms. This chain of nested algorithms leads us to the following principle:

Principle of Equivalence of Hardware and Software: *Any task done by software can also be done using hardware, and any operation performed directly by hardware can be done using software.*¹

A special-purpose computer can be designed to perform any task, such as word processing, budget analysis, or playing a friendly game of Tetris. Accordingly, programs can be written to carry out the functions of special-purpose computers, such as the embedded systems situated in your car or microwave. There are times when a simple embedded system gives us much better performance than a complicated computer program, and there are times when a program is the preferred approach. The Principle of Equivalence of Hardware and Software tells us that we have a choice. Our knowledge of computer organization and architecture will help us to make the best choice.

We begin our discussion of computer hardware by looking at the components necessary to build a computing system. At the most basic level, a computer is a device consisting of three pieces:

1. A processor to interpret and execute programs
2. A memory to store both data and programs
3. A mechanism for transferring data to and from the outside world

¹What this principle does not address is the speed with which the equivalent tasks are carried out. Hardware implementations are almost always faster.

We discuss these three components in detail as they relate to computer hardware in the following chapters.

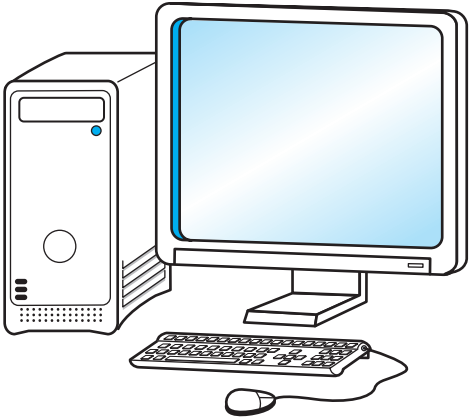
Once you understand computers in terms of their component parts, you should be able to understand what a system is doing at all times and how you could change its behavior if so desired. You might even feel like you have a few things in common with it. This idea is not as far-fetched as it appears. Consider how a student sitting in class exhibits the three components of a computer: The student's brain is the processor, the notes being taken represent the memory, and the pencil or pen used to take notes is the I/O mechanism. But keep in mind that your abilities far surpass those of any computer in the world today, or any that can be built in the foreseeable future.

1.3 AN EXAMPLE SYSTEM: WADING THROUGH THE JARGON

This text will introduce you to some of the vocabulary that is specific to computers. This jargon can be confusing, imprecise, and intimidating. We believe that with a little explanation, we can clear the fog.

For the sake of discussion, we have provided a facsimile computer advertisement (see Figure 1.1). The ad is typical of many in that it bombards the reader with phrases such as “32GB DDR3 SDRAM,” “PCIe sound card,” and “128KB L1 cache.” Without having a handle on such terminology, you would be hard-pressed to know whether the stated system is a wise buy, or even whether the

FOR SALE: OBSOLETE COMPUTER – CHEAP! CHEAP! CHEAP!



- Intel i7 Quad Core, 3.9GHz
- 1600MHz 32GB DDR3 SDRAM
- 128KB L1 cache, 2MB L2 cache
- 1TB SATA hard drive (7200 RPM)
- 10 USB ports, 1 serial port, 4 PCI expansion slots (1 PCI, 1 PCIx16, 2 PCIx1), Bluetooth, and HDMI
- 24" widescreen LCD monitor, 16:10 aspect ratio, 1920x1200 WUXGA, 300 cd/m², active matrix, 1000:1 (static), 8ms, 24-bit color (16.7 million colors), VGA/DVI input, 2 USB ports
- 16x CD/DVD +/- RW drive
- 1GB PCIe video card
- PCIe sound card
- Integrated 10/100/1000 Ethernet

FIGURE 1.1 A Typical Computer Advertisement

system is able to serve your needs. As we progress through this text, you will learn the concepts behind these terms.

Before we explain the ad, however, we need to discuss something even more basic: the measurement terminology you will encounter throughout your study of computers.

It seems that every field has its own way of measuring things. The computer field is no exception. For computer people to tell each other how big something is, or how fast something is, they must use the same units of measure. The common prefixes used with computers are given in Table 1.1. Back in the 1960s, someone decided that because the powers of 2 were close to the powers of 10, the same prefix names could be used for both. For example, 2^{10} is close to 10^3 , so “kilo” is used to refer to them both. The result has been mass confusion: Does a given prefix refer to a power of 10 or a power of 2? Does a kilo mean 10^3 of something or 2^{10} of something? Although there is no definitive answer to this question, there are accepted “standards of usage.” Power-of-10 prefixes are ordinarily used for power, electrical voltage, frequency (such as computer clock speeds), and multiples of bits (such as data speeds in number of bits per second). If your antiquated modem transmits at 28.8kb/s, then it transmits 28,800 bits per second (or 28.8×10^3). Note the use of the lowercase “k” to mean 10^3 and the lowercase “b” to refer to bits. An uppercase “K” is used to refer to the power-of-2 prefix, or 1024. If a file is 2KB in size, then it is 2×2^{10} or 2048 bytes. Note the uppercase “B” to refer to byte. If a disk holds 1MB, then it holds 2^{20} bytes (or one megabyte) of information.

Not knowing whether specific prefixes refer to powers of 2 or powers of 10 can be very confusing. For this reason, the International Electrotechnical Commission, with help from the National Institute of Standards and Technology, has approved standard names and symbols for binary prefixes to differentiate them from decimal prefixes. Each prefix is derived from the symbols given in Table 1.1 by adding an “i.” For example, 2^{10} has been renamed “kibi” (for kilobinary) and

Prefix	Symbol	Power of 10	Power of 2	Prefix	Symbol	Power of 10	Power of 2
Kilo	K	1 thousand = 10^3	$2^{10} = 1024$	Milli	m	1 thousandth = 10^{-3}	2^{-10}
Mega	M	1 million = 10^6	2^{20}	Micro	μ	1 millionth = 10^{-6}	2^{-20}
Giga	G	1 billion = 10^9	2^{30}	Nano	n	1 billionth = 10^{-9}	2^{-30}
Tera	T	1 trillion = 10^{12}	2^{40}	Pico	p	1 trillionth = 10^{-12}	2^{-40}
Peta	P	1 quadrillion = 10^{15}	2^{50}	Femto	f	1 quadrillionth = 10^{-15}	2^{-50}
Exa	E	1 quintillion = 10^{18}	2^{60}	Atto	a	1 quintillionth = 10^{-18}	2^{-60}
Zetta	Z	1 sextillion = 10^{21}	2^{70}	Zepto	z	1 sextillionth = 10^{-21}	2^{-70}
Yotta	Y	1 septillion = 10^{24}	2^{80}	Yocto	y	1 septillionth = 10^{-24}	2^{-80}

TABLE 1.1 Common Prefixes Associated with Computer Organization and Architecture

is represented by the symbol Ki. Similarly, 2^{20} is mebi, or Mi, followed by gibi (Gi), tebi (Ti), pebi (Pi), exbi (Ei), and so on. Thus, the term *mebibyte*, which means 2^{20} bytes, replaces what we traditionally call a megabyte.

There has been limited adoption of these new prefixes. This is unfortunate because, as a computer user, it is important to understand the true meaning of these prefixes. A kilobyte (1KB) of memory is *typically* 1024 bytes of memory rather than 1000 bytes of memory. However, a 1GB disk drive might actually be 1 billion bytes instead of 2^{30} (which means you are getting less storage than you think). All 3.5" floppy disks are described as storing 1.44MB of data when in fact they store 1440KB (or $1440 \times 2^{10} = 1474560$ bytes). You should always read the manufacturer's fine print just to make sure you know exactly what 1K, 1KB, or 1G represents. See the sidebar "When a Gigabyte Isn't Quite . . ." for a good example of why this is so important.

Who Uses Zettabytes and Yottabytes Anyway?

The National Security Agency (NSA), an intelligence-gathering organization in the United States, announced that its new Intelligence Community Comprehensive National Cybersecurity Initiative Data Center, in Bluffdale, Utah, was set to open in October 2013. Approximately 100,000 square feet of the structure is utilized for the data center. Whereas the remaining 900,000+ square feet houses technical support and administration. The new data center will help the NSA monitor the vast volume of data traffic on the Internet.

It is estimated that the NSA collects roughly 2 million gigabytes of data every hour, 24 hours a day, seven days a week. This data includes foreign and domestic emails, cell phone calls, Internet searches, various purchases, and other forms of digital data. The computer responsible for analyzing this data for the new data center is the Titan supercomputer, a water-cooled machine capable of operating at 100 petaflops (or 100,000 trillion calculations each second). The PRISM (Planning Tool for Resource Integration, Synchronization, and Management) surveillance program will gather, process, and track all collected data.

Although we tend to think in terms of gigabytes and terabytes when buying storage for our personal computers and other devices, the NSA's data center storage capacity will be measured in zettabytes (with many hypothesizing that storage will be in thousands of zettabytes, or yottabytes). To put this in perspective, in a 2003 study done at the University of California (UC) Berkeley, it was estimated that the amount of new data created in 2002 was roughly 5EB. An earlier study by UC Berkeley estimated that by the end of 1999, the sum of all information, including audio, video, and text, created by humankind was approximately 12EB of data. In 2006, the combined storage space of every computer hard drive in the world was estimated at 160EB; in 2009, the Internet as a whole was estimated to contain roughly 500 total exabytes, or a half zettabyte, of data. Cisco, a U.S. computer network hardware manufacturer, has estimated that by 2016, the total volume of data on the global internet will be 1.3ZB, and Seagate Technology, an American manufacturer of hard drives, has estimated that the total storage capacity demand will reach 7ZB in 2020.

The NSA is not the only organization dealing with information that must be measured in numbers of bytes beyond the typical “giga” and “tera.” It is estimated that Facebook collects 500TB of new material per day; YouTube observes roughly 1TB of new video information every four minutes; the CERN Large Hadron Collider generates 1PB of data per second; and the sensors on a single, new Boeing jet engine produce 20TB of data every hour. Although not all of the aforementioned examples require permanent storage of the data they create/handle, these examples nonetheless provide evidence of the remarkable quantity of data we deal with every day. This tremendous volume of information is what prompted the IBM Corporation, in 2011, to develop and announce its new 120-PB hard drive, a storage cluster consisting of 200,000 conventional hard drives harnessed to work together as a single unit. If you plugged your MP3 player into this drive, you would have roughly two billion hours of music!

In this era of smartphones, tablets, Cloud computing, and other electronic devices, we will continue to hear people talking about petabytes, exabytes, and zettabytes (and, in the case of the NSA, even yottabytes). However, if we outgrow yottabytes, what then? In an effort to keep up with the astronomical growth of information and to refer to even bigger volumes of data, the next generation of prefixes will most likely include the terms *brontobyte* for 10^{27} and *gegobyte* for 10^{30} (although some argue for *geobyte* and *geopbyte* as the prefixes for the latter). Although these are not yet universally accepted international prefix units, if history is any indication, we will need them sooner rather than later.

When a Gigabyte Isn't Quite . . .

Purchasing a new array of disk drives should be a relatively straightforward process once you determine your technical requirements (e.g., disk transfer rate, interface type, etc.). From here, you should be able to make your decision based on a simple price/capacity ratio, such as dollars per gigabyte, and then you'll be done. Well, not so fast.

The first boulder in the path of a straightforward analysis is that you must make sure that the drives you are comparing all express their capacities either in formatted or unformatted bytes. As much as 16% of drive space is consumed during the formatting process. (Some vendors give this number as “usable capacity.”) Naturally, the price–capacity ratio looks much better when unformatted bytes are used, although you are most interested in knowing the amount of *usable* space a disk provides.

Your next obstacle is to make sure that the same radix is used when comparing disk sizes. It is increasingly common for disk capacities to be given in base 10 rather than base 2. Thus, a “1GB” disk drive has a capacity of $10^9 = 1,000,000,000$ bytes, rather than $2^{30} = 1,073,741,824$ bytes—a reduction of about 7%. This can make a huge difference when purchasing multigigabyte enterprise-class storage systems.

As a concrete example, suppose you are considering purchasing a disk array from one of two leading manufacturers. Manufacturer x advertises an array of 12 250GB

disks for \$20,000. Manufacturer y is offering an array of 12 212.5GB disks for \$21,000. All other things being equal, the cost ratio overwhelmingly favors Manufacturer x:

Manufacturer x: $\$20,000 \div (12 \times 250\text{GB}) \cong \6.67 per GB

Manufacturer y: $\$21,000 \div (12 \times 212.5\text{GB}) \cong \8.24 per GB

Being a little suspicious, you make a few telephone calls and learn that Manufacturer x is citing capacities using unformatted base 10 gigabytes and Manufacturer y is using formatted base 2 gigabytes. These facts cast the problem in an entirely different light: To start with, Manufacturer x's disks aren't really 250GB in the way that we usually think of gigabytes. Instead, they are about 232.8 base 2 gigabytes. After formatting, the number reduces even more to about 197.9GB. So the real cost ratios are, in fact:

Manufacturer x: $\$20,000 \div (12 \times 197.9\text{GB}) \cong \8.42 per GB

Manufacturer y: $\$21,000 \div (12 \times 212.5\text{GB}) \cong \8.24 per GB

Indeed, some vendors are scrupulously honest in disclosing the capabilities of their equipment. Unfortunately, others reveal the facts only under direct questioning. Your job as an educated professional is to ask the right questions.

When we want to talk about how fast something is, we speak in terms of fractions of a second—usually thousandths, millionths, billionths, or trillionths. Prefixes for these metrics are given in the right-hand side of Table 1.1. Generally, negative powers refer to powers of 10, not powers of 2. For this reason, the new binary prefix standards do not include any new names for the negative powers. Notice that the fractional prefixes have exponents that are the reciprocal of the prefixes on the left side of the table. Therefore, if someone says to you that an operation requires a microsecond to complete, you should also understand that a million of those operations could take place in one second. When you need to talk about how many of these things happen in a second, you would use the prefix *mega-*. When you need to talk about how fast the operations are performed, you would use the prefix *micro-*.

Now to explain the ad. The microprocessor in the ad is an Intel i7 Quad Core processor (which means it is essentially four processors) and belongs to a category of processors known as multicore processors (Section 1.10 contains more information on multicore processors). This particular processor runs at 3.9GHz. Every computer system contains a clock that keeps the system synchronized. The clock sends electrical pulses simultaneously to all main components, ensuring that data and instructions will be where they're supposed to be, when they're supposed to be there. The number of pulsations emitted each second by the clock is its frequency. Clock frequencies are measured in cycles per second, or **hertz**. If computer system clocks generate millions of pulses per second, we say that they operate in the **megahertz (MHz)** range. Most computers today operate in the **gigahertz (GHz)** range, generating billions of

pulses per second. And because nothing much gets done in a computer system without microprocessor involvement, the frequency rating of the microprocessor is crucial to overall system speed. The microprocessor of the system in our advertisement operates at 3.9 billion cycles per second, so the seller says that it runs at 3.9GHz.

The fact that this microprocessor runs at 3.9GHz, however, doesn't necessarily mean that it can execute 3.9 billion instructions every second or, equivalently, that every instruction requires 0.039 nanoseconds to execute. Later in this text, you will see that each computer instruction requires a fixed number of cycles to execute. Some instructions require one clock cycle; however, most instructions require more than one. The number of instructions per second that a microprocessor can actually execute is *proportionate* to its clock speed. The number of clock cycles required to carry out a particular machine instruction is a function of both the machine's organization and its architecture.

The next thing we see in the ad is "1600MHz 32GB DDR3 SDRAM." The 1600MHz refers to the speed of the system **bus**, which is a group of wires that moves data and instructions to various places within the computer. Like the microprocessor, the speed of the bus is also measured in MHz or GHz. Many computers have a special local bus for data that supports very fast transfer speeds (such as those required by video). This local bus is a high-speed pathway that connects memory directly to the processor. Bus speed ultimately sets the upper limit on the system's information-carrying capability.

The system in our advertisement also boasts a memory capacity of 32 gigabytes (GB), or about 32 billion characters. Memory capacity not only determines the size of the programs you can run, but also how many programs you can run at the same time without bogging down the system. Your application or operating system manufacturer will usually recommend how much memory you'll need to run its products. (Sometimes these recommendations can be hilariously conservative, so be careful whom you believe!)

In addition to memory size, our advertised system provides us with a memory type, **SDRAM**, short for **synchronous dynamic random access memory**. SDRAM is much faster than conventional (nonsynchronous) memory because it can synchronize itself with a microprocessor's bus. The system in our ad has **DDR3 SDRAM**, or **double data rate** type three SDRAM (for more information on the different types of memory, see Chapter 6).

A Look Inside a Computer

Have you ever wondered what the inside of a computer really looks like? The example computer described in this section gives a good overview of the components of a modern PC. However, opening a computer and attempting to find and identify the various pieces can be frustrating, even if you are familiar with the components and their functions.

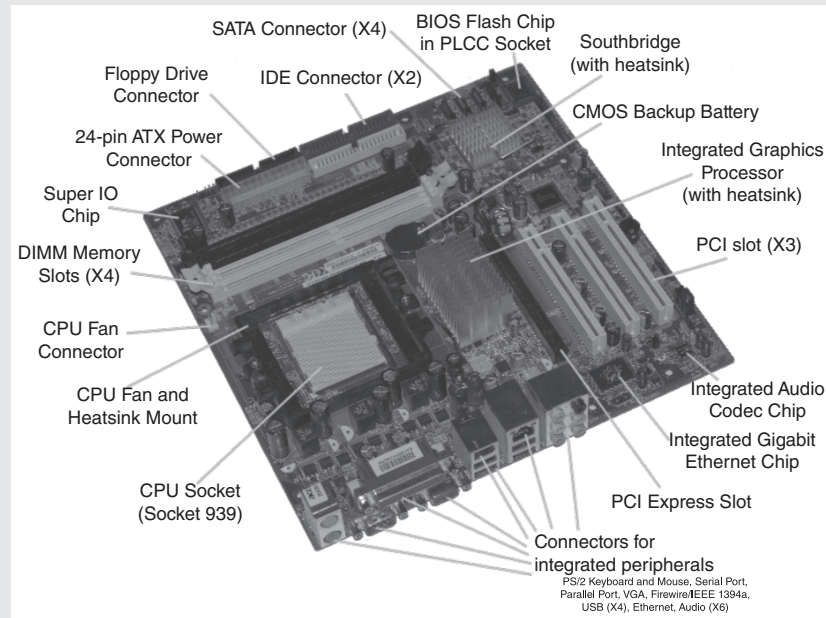


Photo courtesy of Moxfyre at en.wikipedia (from http://commons.wikimedia.org/wiki/File:Acer_E360_Socket_939_motherboard_by_Foxconn.svg).

If you remove the cover on your computer, you will no doubt first notice a big metal box with a fan attached. This is the power supply. You will also see various drives, including a hard drive and a DVD drive (or perhaps an older floppy or CD drive). There are many integrated circuits—small, black rectangular boxes with legs attached. You will also notice electrical pathways, or buses, in the system. There are printed circuit boards (expansion cards) that plug into sockets on the motherboard, the large board at the bottom of a standard desktop PC or on the side of a PC configured as a tower or mini-tower. The motherboard is the printed circuit board that connects all the components in the computer, including the CPU, and RAM and ROM, as well as an assortment of other essential components. The components on the motherboard tend to be the most difficult to identify. Above you see an Acer E360 motherboard with the more important components labeled.

The Southbridge, an integrated circuit that controls the hard disk and I/O (including sound and video cards), is a hub that connects slower I/O devices to the system bus. These devices connect via the I/O ports at the bottom of the board. The PCI slots allow for expansion boards belonging to various PCI devices. This motherboard also has PS/2 and Firewire connectors. It has serial and parallel ports, in addition to four USB ports. This motherboard has two IDE connector slots, four SATA connector slots, and one floppy disk controller. The super I/O chip is a type of I/O controller that controls the floppy disk, both the parallel and serial ports, and the keyboard and mouse. The motherboard also has an integrated audio chip, as well

as an integrated Ethernet chip and an integrated graphics processor. There are four RAM memory banks. There is no processor currently plugged into this motherboard, but we see the socket where the CPU is to be placed. All computers have an internal battery, as seen in the top middle of the picture. The power supply plugs into the power connector. The BIOS flash chip contains the instructions in ROM that your computer uses when it is first powered up.

A note of caution regarding looking inside the box: There are many safety issues, for both you and your computer, involved with removing the cover. There are many things you can do to minimize the risks. First and foremost, make sure the computer is turned off. Leaving it plugged in is often preferred, as this offers a path for static electricity. Before opening your computer and touching anything inside, you should make sure you are properly grounded so static electricity will not damage any components. Many of the edges, both on the cover and on the circuit boards, can be sharp, so take care when handling the various pieces. Trying to jam misaligned cards into sockets can damage both the card and the motherboard, so be careful if you decide to add a new card or remove and reinstall an existing one.

The next line in the ad, “128KB L1 cache, 2MB L2 cache” also describes a type of memory. In Chapter 6, you will learn that no matter how fast a bus is, it still takes “a while” to get data from memory to the processor. To provide even faster access to data, many systems contain a special memory called **cache**. The system in our advertisement has two kinds of cache. Level 1 cache (L1) is a small, fast memory cache that is built into the microprocessor chip and helps speed up access to frequently used data. Level 2 cache (L2) is a collection of fast, built-in memory chips situated between the microprocessor and main memory. Notice that the cache in our system has a capacity of kilobytes (KB), which is much smaller than main memory. In Chapter 6, you will learn how cache works, and that a bigger cache isn’t always better.

On the other hand, everyone agrees that the more fixed disk capacity you have, the better off you are. The advertised system has a 1TB hard drive, an average size by today’s standards. The storage capacity of a fixed (or hard) disk is not the only thing to consider, however. A large disk isn’t very helpful if it is too slow for its host system. The computer in our ad has a hard drive that rotates at 7200 revolutions per minute (RPM). To the knowledgeable reader, this indicates (but does not state outright) that this is a fairly fast drive. Usually, disk speeds are stated in terms of the number of milliseconds required (on average) to access data on the disk, in addition to how fast the disk rotates.

Rotational speed is only one of the determining factors in the overall performance of a disk. The manner in which it connects to—or **interfaces** with—the rest of the system is also important. The advertised system uses a **SATA (serial advanced technology attachment or serial ATA)** disk interface. This is an evolutionary storage interface that has replaced **IDE**, or **integrated drive electronics**. Another common interface is **EIDE, enhanced integrated drive electronics**, a cost-effective hardware interface alternative for mass storage devices. EIDE

contains special circuits that allow it to enhance a computer's connectivity, speed, and memory capability. Most ATA, IDE, and EIDE systems share the main system bus with the processor and memory, so the movement of data to and from the disk is also dependent on the speed of the system bus.

Whereas the system bus is responsible for all data movement internal to the computer, **ports** allow movement of data to and from devices external to the computer. Our ad speaks of two different ports with the line, "10 USB ports, 1 serial port." Serial ports transfer data by sending a series of electrical pulses across one or two data lines. Another type of port some computers have is a parallel port. Parallel ports use at least eight data lines, which are energized simultaneously to transmit data. Many new computers no longer come with serial or parallel ports, but instead have only USB ports. USB (**universal serial bus**) is a popular external bus that supports **Plug-and-Play** installation (the ability to configure devices automatically) as well as **hot plugging** (the ability to add and remove devices while the computer is running).

Expansion slots are openings on the motherboard where various boards can be plugged in to add new capabilities to a computer. These slots can be used for such things as additional memory, video cards, sound cards, network cards, and modems. Some systems augment their main bus with dedicated I/O buses using these expansion slots. **Peripheral Component Interconnect (PCI)** is one such I/O bus standard that supports the connection of multiple peripheral devices. PCI, developed by the Intel Corporation, operates at high speeds and also supports Plug-and-Play.

PCI is an older standard (it has been around since 1993) and was superseded by PCI-x in 2004. PCI-x basically doubled the bandwidth of regular PCI. Both PCI and PCI-x are parallel in operation. In 2004, PCI express (PCIe) replaced PCI-x. PCIe operates in serial and is currently the standard in today's computers. In the ad, we see the computer has 1 PCI slot, 1 PCI x 16 slot, and 2 PCI x 1 slots. This computer also has Bluetooth (a wireless technology allowing the transfer of information over short distances) and an HDMI port (High-Definition Multimedia Interface, used to transmit audio and video).

PCIe has not only superseded PCI and PCI-x, but in the graphics world, it has also progressively replaced the **AGP (accelerated graphics port)** graphics interface designed by Intel specifically for 3D graphics. The computer in our ad has a PCIe video card with 1GB of memory. The memory is used by a special **graphics processing unit** on the card. This processor is responsible for performing the necessary calculations to render the graphics so the main processor of the computer is not required to do so. This computer also has a PCIe sound card; a sound card contains components needed by the system's stereo speakers and microphone.

In addition to telling us about the ports and expansion slots in the advertised system, the ad supplies us with information on an **LCD (liquid crystal display)** monitor, or "flat panel" display. Monitors have little to do with the speed or efficiency of a computer system, but they have great bearing on the comfort of the user. This LCD monitor has the following specifications: 24", 1920 × 1200 WUXGA, 300 cd/m², active matrix, 1000:1 (static), 8ms, 24-bit color (16.7 million colors), VGA/DVI input, and 2USB ports. LCDs use a liquid crystal material sandwiched

between two pieces of polarized glass. Electric currents cause the crystals to move around, allowing differing levels of backlighting to pass through, creating the text, colors, and pictures that appear on the screen. This is done by turning on/off different **pixels**, small “picture elements” or dots on the screen. Monitors typically have millions of pixels, arranged in rows and columns. This monitor has 1920×1200 (more than a million) pixels.

Most LCDs manufactured today utilize active matrix technology. Whereas passive technology is reserved for smaller devices such as calculators and clocks. **Active matrix** technology uses one transistor per pixel; **passive matrix** technology uses transistors that activate entire rows and columns. Although passive technology is less costly, active technology renders a better image because it drives each pixel independently.

The LCD monitor in the ad is 24", measured diagonally. This measurement affects the **aspect ratio** of the monitor—the ratio of horizontal pixels to vertical pixels that the monitor can display. Traditionally, this ratio was 4:3, but newer widescreen monitors use ratios of 16:10 or 16:9. Ultra-wide monitors use a higher ratio, around 3:1 or 2:1.

When discussing resolution and LCDs, it is important to note that LCDs have a **native resolution**; this means LCDs are designed for a specific resolution (generally given in horizontal pixels by vertical pixels). Although you can change the resolution, the image quality typically suffers. Resolutions and aspect ratios are often paired. When listing resolutions for LCDs, manufacturers often use the following abbreviations: XGA (extended graphics array); XGA+ (extended graphics array plus); SXGA (super XGA); UXGA (ultra XGA); W prefix (wide); and WVA (wide viewing angle). The viewing angle specifies an angle, in degrees, that indicates at which angle a user can still see the image on the screen; common angles range from 120 to 170 degrees. Some examples of standard 4:3 native resolutions include XGA (1024×768), SXGA (1280×1024), SXGA+ (1400×1050), and UXGA (1600×1200). Common 16:9 and 16:10 resolutions include WXGA (1280×800), WXGA+ (1440×900), WSXGA+ (1680×1050), and WUXGA (1920×1200).

LCD monitor specifications often list a **response time**, which indicates the rate at which the pixels can change colors. If this rate is too slow, ghosting and blurring can occur. The LCD monitor in the ad has a response time of 8ms. Originally, response rates measured the time to go from black to white and back to black. Many manufacturers now list the response time for gray-to-gray transitions (which is generally faster). Because they typically do not specify which transition has been measured, it is very difficult to compare monitors. One manufacturer may specify a response time of 2ms for a monitor (and it measures gray-to-gray), while another manufacturer may specify a response rate of 5ms for its monitor (and it measures black-to-white-to-black). In reality, the monitor with the response rate of 5ms may actually be faster overall.

Continuing with the ad, we see that the LCD monitor has a specification of 300 cd/m^2 , which is the monitor's luminance. **Luminance** (or image brightness) is a measure of the amount of light an LCD monitor emits. This measure is typically given in candelas per square meter (cd/m^2). When purchasing a monitor,

the brightness level should be at least 250 (the higher the better); the average for computer monitors is from 200 to 300 cd/m^2 . Luminance affects how easy a monitor is to read, particularly in low light situations.

Whereas luminance measures the brightness, the **contrast ratio** measures the difference in intensity between bright whites and dark blacks. Contrast ratios can be **static** (the ratio of the brightest point on the monitor to the darkest point on the monitor that can be produced at a given instant in time) or **dynamic** (the ratio of the darkest point in one image to the lightest point in another image produced at a separate point in time). Static specifications are typically preferred. A low static ratio (such as 300:1) makes it more difficult to discern shades; a good static ratio is 500:1 (with ranges from 400:1 to 3000:1). The monitor in the ad has a static contrast ratio of 1000:1. LCD monitors can have dynamic ratios of 12,000,000:1 and higher, but a higher dynamic number does not necessarily mean the monitor is better than a monitor with a much lower static ratio.

The next specification given for the LCD monitor in the ad is its **color depth**. This number reflects the number of colors that can be displayed on the screen at one time. Common depths are 8-bit, 16-bit, 24-bit, and 32-bit. The LCD monitor in our ad can display 2^{24} , or roughly 16.7 million colors.

LCD monitors also have many optional features. Some have integrated USB ports (as in this ad) and/or speakers. Many are HDCP (high bandwidth digital content protection) compliant (which means you can watch HDCP-encrypted materials, such as Blu-ray discs). LCD monitors may also come with both VGA (video graphics array) and DVI (digital video interface) connections (as seen in the ad). VGA sends analog signals to the monitor from the computer, which requires digital-to-analog conversion; DVI is already digital in format and requires no conversion, resulting in a cleaner signal and crisper image. Although an LCD monitor typically provides better images using a DVI connection, having both connectors allows one to use an LCD with existing system components.

Now that we have discussed how an LCD monitor works and we understand the concept of a pixel, let's go back and discuss graphics cards (also called video cards) in more detail. With millions of pixels on the screen, it is quite challenging to determine which ones should be off and which ones should be on (and in what color). The job of the graphics card is to input the binary data from your computer and "translate" it into signals to control all pixels on the monitor; the graphics card therefore acts as a "middleman" between the computer's processor and monitor. As mentioned previously, some computers have integrated graphics, which means the computer's processor is responsible for doing this translation, causing a large workload on this processor; therefore, many computers have slots for graphics cards, allowing the processor on the graphics card (called a **graphics processing unit**, or **GPU**) to perform this translation instead.

The GPU is no ordinary processor; it is designed to most efficiently perform the complex calculations required for image rendering and contains special programs allowing it to perform this task more effectively. Graphics cards typically contain their own dedicated RAM used to hold temporary results and information, including the location and color for each pixel on the screen. A **frame buffer** (part of

this RAM) is used to store rendered images until these images are intended to be displayed. The memory on a graphics card connects to a **digital-to-analog converter** (DAC), a device that converts a binary image to analog signals that a monitor can understand and sends them via a cable to the monitor. Most graphics cards today have two types of monitor connections: DVI for LCD screens and VGA for the older CRT (cathode ray tube) screens.

Most graphics cards are plugged into slots in computer motherboards, so are thus powered by the computers themselves. However, some are very powerful and actually require a connection directly to a computer's power supply. These high-end graphics cards are typically found in computers that deal with image-intensive applications, such as video editing and high-end gaming.

Continuing with the ad, we see that the advertised system has a 16x DVD +/- RW drive. This means we can read and write to DVDs and CDs. "16x" is a measure of the drive speed and measures how quickly the drive can read and write. DVDs and CDs are discussed in more detail in Chapter 7.

Computers are more useful if they can communicate with the outside world. One way to communicate is to employ an Internet service provider and a modem. There is no mention of a modem for the computer in our ad, as many desktop owners use external modems provided by their Internet service provider (phone modem, cable modem, satellite modem, etc). However, both USB and PCI modems are available that allow you to connect your computer to the Internet using the phone line; many of these also allow you to use your computer as a fax machine. I/O and I/O buses in general are discussed in Chapter 7.

A computer can also connect directly to a network. Networking allows computers to share files and peripheral devices. Computers can connect to a network via either a wired or a wireless technology. Wired computers use **Ethernet** technology, an international standard networking technology for wired networks, and there are two options for the connection. The first is to use a **network interface card (NIC)**, which connects to the motherboard via a PCI slot. NICs typically support 10/100 Ethernet (both Ethernet at a speed of 10Mbps and fast Ethernet at a speed of 100Mbps) or 10/100/1000 (which adds Ethernet at 1,000Mbps). Another option for wired network capability is integrated Ethernet, which means that the motherboard itself contains all necessary components to support 10/100 Ethernet; thus no PCI slot is required. Wireless networking has the same two options. Wireless NICs are available from a multitude of vendors and are available for both desktops and laptops. For installation in desktop machines, you need an internal card that will most likely have a small antenna. Laptops usually use an expansion (PCMCIA) slot for the wireless network card, and vendors have started to integrate the antenna into the back of the case behind the screen. Integrated wireless (such as that found in the Intel Centrino mobile technology) eliminates the hassle of cables and cards. The system in our ad employs integrated Ethernet. Note that many new computers may have integrated graphics and/or integrated sound in addition to integrated Ethernet.

Although we cannot delve into all of the brand-specific components available, after completing this text, you should understand the concept of how most computer systems operate. This understanding is important for casual users

as well as experienced programmers. As a user, you need to be aware of the strengths and limitations of your computer system so you can make informed decisions about applications and thus use your system more effectively. As a programmer, you need to understand exactly how your system hardware functions so you can write effective and efficient programs. For example, something as simple as the algorithm your hardware uses to map main memory to cache and the method used for memory interleaving can have a tremendous effect on your decision to access array elements in row versus column-major order.

Throughout this text, we investigate both large and small computers. Large computers include mainframes, enterprise-class servers, and supercomputers. Small computers include personal systems, workstations, and handheld devices. We will show that regardless of whether they carry out routine chores or perform sophisticated scientific tasks, the components of these systems are very similar. We also visit some architectures that lie outside what is now the mainstream of computing. We hope that the knowledge you gain from this text will ultimately serve as a springboard for your continuing studies the vast and exciting fields of computer organization and architecture.

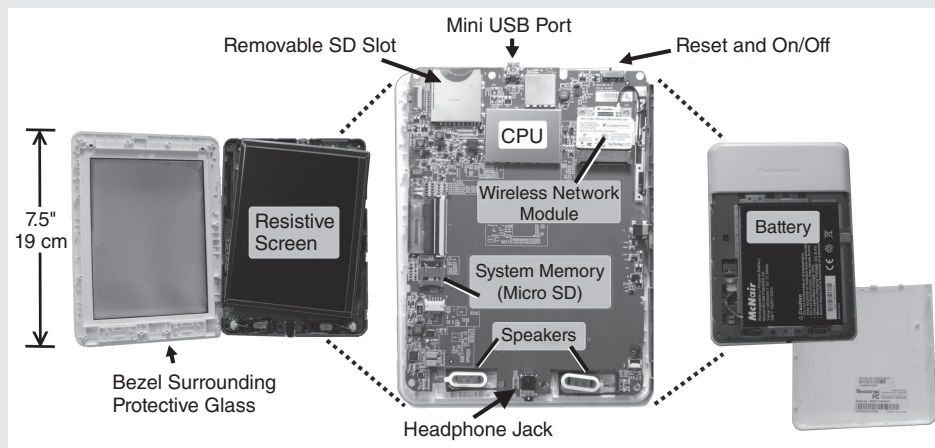
Tablet Computers

Ken Olsen, the founder of Digital Equipment Corporation, has been unfairly ridiculed for saying “There is no reason for any individual to have a computer in his home.” He made this statement in 1977 when the word, *computer*, evoked a vision of the type of machine made by his company: refrigerator-sized behemoths that cost a fortune and required highly skilled personnel to operate. One might safely say that no one—except perhaps a computer engineer—ever had such a machine in his or her home.

As already discussed, the “personal computing” wave that began in the 1980s erupted in the 1990s with the establishment of the World Wide Web. By 2010, decennial census data reported that 68% of U.S. households claimed to have a personal computer. There is, however, some evidence that this trend has peaked and is now in decline, owing principally to the widespread use of smartphones and tablet computers. According to some estimates, as many as 65% of Internet users in the United States connect exclusively via mobile platforms. The key to this trend is certainly the enchanting usability of these devices.

We hardly need the power of a desktop computer to surf the Web, read email, or listen to music. Much more economical and lightweight, tablet computers give us exactly what we need in an easy-to-use package. With its booklike form, one is tempted to claim that a tablet constitutes the perfect “portable computer.”

The figure on the next page shows a disassembled Pandigital Novel tablet computer. We have labeled several items common to all tablets. The mini USB port provides access to internal storage and the removable SD card. Nearly all tablets provide Wi-Fi connection to the Internet, with some also supporting 2G, 3G, and 4G cellular protocols. Battery life can be as much as 14 hours for the most efficient high-end tablet computers. Unlike the Pandigital, most tablets include at least one camera for still photography and live video.



A Disassembled Tablet Computer

Courtesy of Julia Lobur.

A touchscreen dominates the real estate of all portable devices. For consumer tablets and phones, touchscreens come in two general types: resistive and capacitive. **Resistive** touchscreens respond to the pressure of a finger or a stylus. **Capacitive** touchscreens react to the electrical properties of the human skin. Resistive screens are less sensitive than capacitive screens, but they provide higher resolution. Unlike resistive screens, capacitive screens support multitouch, which is the ability to detect the simultaneous press of two or more fingers.

Military and medical computer touchscreens are necessarily more durable than those intended for the consumer market. Two different technologies, **surface acoustic wave touch sense** and **infrared touch sense**, respectively, send ultrasonic and infrared waves across the surface of a ruggedized touchscreen. The matrix of waves is broken when a finger comes in contact with the surface of the screen.

Because of its high efficiency, cell phone CPU technology has been adapted for use in the tablet platform. The mobile computing space has been dominated by ARM chips, although Intel and AMD have been gaining market share. Operating systems for these devices include variants of Android by Google and iOS by Apple. Microsoft's Surface tablets running Windows 8 provide access to the Microsoft Office suite of products.

As tablet computers continue to replace desktop systems, they will also find uses in places where traditional computers—even laptops—are impractical. Thousands of free and inexpensive applications are available for all platforms, thereby increasing demand even further. Educational applications abound. With a size, shape, and weight similar to a paperback book, tablet computers are replacing paper textbooks in some U.S. school districts. Thus, the elusive dream of “a computer for every student” is finally coming true—thanks to the tablet. By 1985, people were already laughing at Olsen's “home computer” assertion. Would perhaps these same people have scoffed if instead he would have predicted a computer in every backpack?

1.4 STANDARDS ORGANIZATIONS

Suppose you decide you'd like to have one of those nifty new LCD widescreen monitors. You figure you can shop around a bit to find the best price. You make a few phone calls, surf the Web, and drive around town until you find the one that gives you the most for your money. From your experience, you know you can buy your monitor anywhere and it will probably work fine on your system. You can make this assumption because computer equipment manufacturers have agreed to comply with connectivity and operational specifications established by a number of government and industry organizations.

Some of these standards-setting organizations are ad hoc trade associations or consortia made up of industry leaders. Manufacturers know that by establishing common guidelines for a particular type of equipment, they can market their products to a wider audience than if they came up with separate—and perhaps incompatible—specifications.

Some standards organizations have formal charters and are recognized internationally as the definitive authority in certain areas of electronics and computers. As you continue your studies in computer organization and architecture, you will encounter specifications formulated by these groups, so you should know something about them.

The **Institute of Electrical and Electronics Engineers (IEEE)** is an organization dedicated to the advancement of the professions of electronic and computer engineering. The IEEE actively promotes the interests of the worldwide engineering community by publishing an array of technical literature. The IEEE also sets standards for various computer components, signaling protocols, and data representation, to name only a few areas of its involvement. The IEEE has a democratic, albeit convoluted, procedure established for the creation of new standards. Its final documents are well respected and usually endure for several years before requiring revision.

The **International Telecommunications Union (ITU)** is based in Geneva, Switzerland. The ITU was formerly known as the *Comité Consultatif International Télégraphique et Téléphonique*, or the International Consultative Committee on Telephony and Telegraphy. As its name implies, the ITU concerns itself with the interoperability of telecommunications systems, including telephone, telegraph, and data communication systems. The telecommunications arm of the ITU, the ITU-T, has established a number of standards that you will encounter in the literature. You will see these standards prefixed by ITU-T or the group's former initials, **CCITT**.

Many countries, including the European Community, have commissioned umbrella organizations to represent their interests in various international groups. The group representing the United States is the **American National Standards Institute (ANSI)**. Great Britain has its **British Standards Institution (BSI)** in addition to having a voice on the **CEN (Comité Européen de Normalisation)**, the European committee for standardization.

The **International Organization for Standardization (ISO)** is the entity that coordinates worldwide standards development, including the activities of ANSI

with BSI, among others. ISO is not an acronym, but derives from the Greek word, *isos*, meaning “equal.” The ISO consists of more than 2800 technical committees, each of which is charged with some global standardization issue. Its interests range from the behavior of photographic film to the pitch of screw threads to the complex world of computer engineering. The proliferation of global trade has been facilitated by the ISO. Today, the ISO touches virtually every aspect of our lives.

Throughout this text, we mention official standards designations where appropriate. Definitive information concerning many of these standards can be found in excruciating detail on the website of the organization responsible for establishing the standard cited. As an added bonus, many standards contain “normative” and informative references, which provide background information in areas related to the standard.

1.5 HISTORICAL DEVELOPMENT

During their 60-year life span, computers have become the perfect example of modern convenience. Living memory is strained to recall the days of steno pools, carbon paper, and mimeograph machines. It sometimes seems that these magical computing machines were developed instantaneously in the form that we now know them. But the developmental path of computers is paved with accidental discovery, commercial coercion, and whimsical fancy. And occasionally computers have even improved through the application of solid engineering practices! Despite all the twists, turns, and technological dead ends, computers have evolved at a pace that defies comprehension. We can fully appreciate where we are today only when we have seen where we’ve come from.

In the sections that follow, we divide the evolution of computers into generations, each generation being defined by the technology used to build the machine. We have provided approximate dates for each generation for reference purposes only. You will find little agreement among experts as to the exact starting and ending times of each technological epoch.

Every invention reflects the time in which it was made, so one might wonder whether it would have been called a computer if it had been invented in the late 1990s. How much computation do we actually see pouring from the mysterious boxes perched on or beside our desks? Until recently, computers served us only by performing mind-bending mathematical manipulations. No longer limited to white-jacketed scientists, today’s computers help us to write documents, keep in touch with loved ones across the globe, and do our shopping chores. Modern business computers spend only a minuscule part of their time performing accounting calculations. Their main purpose is to provide users with a bounty of strategic information for competitive advantage. Has the word *computer* now become a misnomer? An anachronism? What, then, should we call them, if not computers?

We cannot present the complete history of computing in a few pages. Entire texts have been written on this subject and even they leave their readers wanting

more detail. If we have piqued your interest, we refer you to some of the books cited in the list of references at the end of this chapter.

1.5.1 Generation Zero: Mechanical Calculating Machines (1642–1945)

Prior to the 1500s, a typical European businessperson used an abacus for calculations and recorded the result of his ciphering in Roman numerals. After the decimal numbering system finally replaced Roman numerals, a number of people invented devices to make decimal calculations even faster and more accurate. Wilhelm Schickard (1592–1635) has been credited with the invention of the first mechanical calculator, the Calculating Clock (exact date unknown). This device was able to add and subtract numbers containing as many as six digits. In 1642, Blaise Pascal (1623–1662) developed a mechanical calculator called the Pascaline to help his father with his tax work. The Pascaline could do addition with carry and subtraction. It was probably the first mechanical adding device actually used for a practical purpose. In fact, the Pascaline was so well conceived that its basic design was still being used at the beginning of the twentieth century, as evidenced by the Lightning Portable Adder in 1908 and the Addometer in 1920. Gottfried Wilhelm von Leibniz (1646–1716), a noted mathematician, invented a calculator known as the Stepped Reckoner that could add, subtract, multiply, and divide. None of these devices could be programmed or had memory. They required manual intervention throughout each step of their calculations.

Although machines like the Pascaline were used into the twentieth century, new calculator designs began to emerge in the nineteenth century. One of the most ambitious of these new designs was the Difference Engine by Charles Babbage (1791–1871). Some people refer to Babbage as “the father of computing.” By all accounts, he was an eccentric genius who brought us, among other things, the skeleton key and the “cow catcher,” a device intended to push cows and other movable obstructions out of the way of locomotives.

Babbage built his Difference Engine in 1822. The Difference Engine got its name because it used a calculating technique called the **method of differences**. The machine was designed to mechanize the solution of polynomial functions and was actually a calculator, not a computer. Babbage also designed a general-purpose machine in 1833 called the Analytical Engine. Although Babbage died before he could build it, the Analytical Engine was designed to be more versatile than his earlier Difference Engine. The Analytical Engine would have been capable of performing any mathematical operation. The Analytical Engine included many of the components associated with modern computers: an arithmetic processing unit to perform calculations (Babbage referred to this as the **mill**), a memory (the **store**), and input and output devices. Babbage also included a conditional branching operation where the next instruction to be performed was determined by the result of the previous operation. Ada, Countess of Lovelace and daughter of poet Lord Byron, suggested that Babbage write a plan for how the machine would calculate numbers. This is regarded as the first computer program, and Ada is considered to be the first computer programmer. It is also rumored that she suggested the use of the binary number system rather than the decimal number system to store data.

A perennial problem facing machine designers has been how to get data into the machine. Babbage designed the Analytical Engine to use a type of punched card for input and programming. Using cards to control the behavior of a machine did not originate with Babbage, but with one of his friends, Joseph-Marie Jacquard (1752–1834). In 1801, Jacquard invented a programmable weaving loom that could produce intricate patterns in cloth. Jacquard gave Babbage a tapestry that had been woven on this loom using more than 10,000 punched cards. To Babbage, it seemed only natural that if a loom could be controlled by cards, then his Analytical Engine could be as well. Ada expressed her delight with this idea, writing, “[T]he Analytical Engine weaves algebraical patterns just as the Jacquard loom weaves flowers and leaves.”

The punched card proved to be the most enduring means of providing input to a computer system. Keyed data input had to wait until fundamental changes were made in how calculating machines were constructed. In the latter half of the nineteenth century, most machines used wheeled mechanisms, which were difficult to integrate with early keyboards because they were levered devices. But levered devices could easily punch cards and wheeled devices could easily read them. So a number of devices were invented to encode and then “tabulate” card-punched data. The most important of the late-nineteenth-century tabulating machines was the one invented by Herman Hollerith (1860–1929). Hollerith’s machine was used for encoding and compiling 1890 census data. This census was completed in record time, thus boosting Hollerith’s finances and the reputation of his invention. Hollerith later founded the company that would become IBM. His 80-column punched card, the **Hollerith card**, was a staple of automated data processing for more than 50 years.

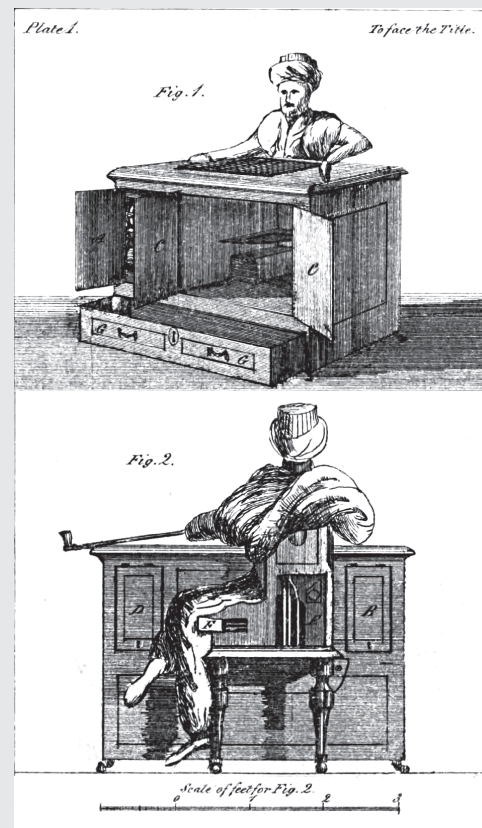
A Pre-Modern “Computer” Hoax

The latter half of the sixteenth century saw the beginnings of the first Industrial Revolution. The spinning jenny allowed one textile worker to do the work of twenty, and steam engines had power equivalent to hundreds of horses. Thus began our enduring fascination with all things mechanical. With the right skills applied to the problems at hand, there seemed no limits to what humankind could do with its machines!

Elaborate clocks began appearing at the beginning of the 1700s. Complex and ornate models graced cathedrals and town halls. These clockworks eventually morphed into mechanical robots called **automata**. Typical models played musical instruments such as flutes and keyboard instruments. In the mid-1700s, the most sublime of these devices entertained royal families across Europe. Some relied on trickery to entertain their audiences. It soon became something of a sport to unravel the chicanery. Empress Marie-Therese of the Austria-Hungarian Empire relied on a wealthy courtier and tinkerer, Wolfgang von Kempelen, to debunk the spectacles on her behalf. One day, following a particularly impressive display, Marie-Therese challenged von Kempelen to build an automaton to surpass all that had ever been brought to her court.

von Kempelen took the challenge, and after several months' work, he delivered a turban-wearing, pipe-smoking, chess-playing automaton. For all appearances, "The Turk" was a formidable opponent for even the best players of the day. As an added touch, the machine contained a set of baffles enabling it to rasp "Échec!" as needed. So impressive was this machine that for 84 years it drew crowds across Europe and the United States.

Of course, as with all similar automata, von Kempelen's Turk relied on trickery to perform its prodigious feat. Despite some astute debunkers correctly deducing how it was done, the secret of the Turk was never divulged: A human chess player was cleverly concealed inside its cabinet. The Turk thus pulled off one of the first and most impressive "computer" hoaxes in the history of technology. It would take another 200 years before a real machine could match the Turk—without the trickery.



The mechanical Turk

Reprinted from Robert Willis, *An attempt to Analyse the Automaton Chess Player of Mr. de Kempelen*. JK Booth, London. 1824.

1.5.2 The First Generation: Vacuum Tube Computers (1945–1953)

Although Babbage is often called the "father of computing," his machines were mechanical, not electrical or electronic. In the 1930s, Konrad Zuse (1910–1995) picked up where Babbage left off, adding electrical technology and other improvements to Babbage's design. Zuse's computer, the Z1, used electromechanical relays instead of Babbage's hand-cranked gears. The Z1 was programmable and had a memory, an arithmetic unit, and a control unit. Because money and resources were scarce in wartime Germany, Zuse used discarded movie film instead of punched cards for input. Although his machine was designed to use vacuum tubes, Zuse, who was building his machine on his own, could not afford the tubes. Thus, the Z1 correctly belongs in the first generation, although it had no tubes.

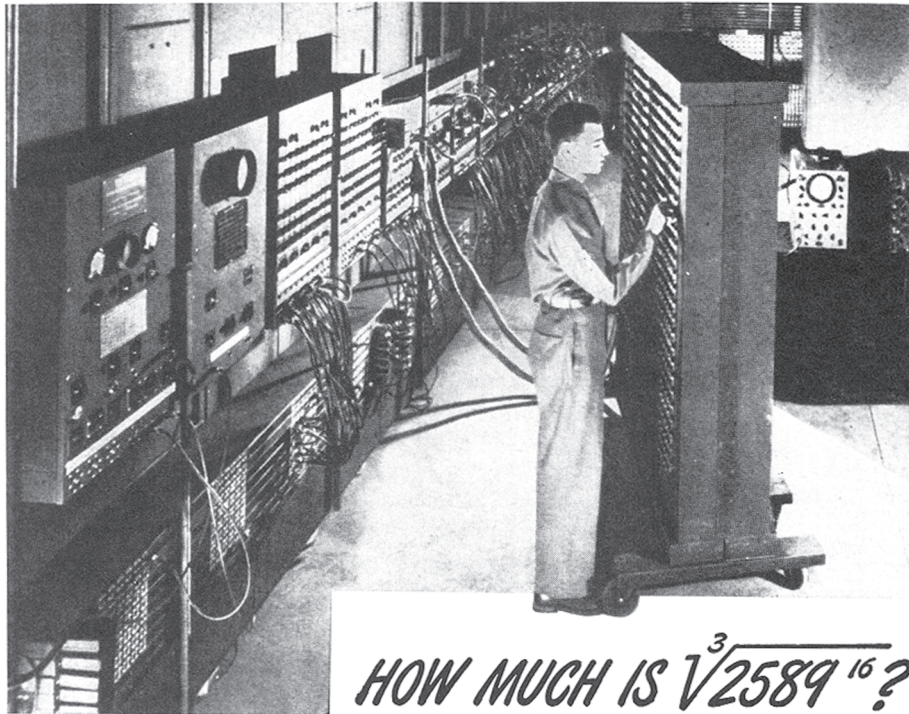
Zuse built the Z1 in his parents' Berlin living room while Germany was at war with most of Europe. Fortunately, he couldn't convince the Nazis to buy his machine. They did not realize the tactical advantage such a device would give them. Allied bombs destroyed all three of Zuse's first systems, the Z1, Z2, and Z3. Zuse's impressive machines could not be refined until after the war and ended up being another "evolutionary dead end" in the history of computers.

Digital computers, as we know them today, are the outcome of work done by a number of people in the 1930s and 1940s. Pascal's basic mechanical calculator was designed and modified simultaneously by many people; the same can be said of the modern electronic computer. Notwithstanding the continual arguments about who was first with what, three people clearly stand out as the inventors of modern computers: John Atanasoff, John Mauchly, and J. Presper Eckert.

John Atanasoff (1904–1995) has been credited with the construction of the first completely electronic computer. The Atanasoff Berry Computer (ABC) was a binary machine built from vacuum tubes. Because this system was built specifically to solve systems of linear equations, we cannot call it a general-purpose computer. There were, however, some features that the ABC had in common with the general-purpose ENIAC (Electronic Numerical Integrator and Computer), which was invented a few years later. These common features caused considerable controversy as to who should be given the credit (and patent rights) for the invention of the electronic digital computer. (The interested reader can find more details on a rather lengthy lawsuit involving Atanasoff and the ABC in Mollenhoff [1988].)

John Mauchly (1907–1980) and J. Presper Eckert (1929–1995) were the two principal inventors of the ENIAC, introduced to the public in 1946. The ENIAC is recognized as the first all-electronic, general-purpose digital computer. This machine used 17,468 vacuum tubes, occupied 1800 square feet of floor space, weighed 30 tons, and consumed 174 kilowatts of power. The ENIAC had a memory capacity of about 1000 information bits (about 20 10-digit decimal numbers) and used punched cards to store data.

John Mauchly's vision for an electronic calculating machine was born from his lifelong interest in predicting the weather mathematically. While a professor of physics at Ursinus College near Philadelphia, Mauchly engaged dozens of adding machines and student operators to crunch mounds of data that he believed would reveal mathematical relationships behind weather patterns. He felt that if he could have only a little more computational power, he could reach the goal that seemed just beyond his grasp. Pursuant to the Allied war effort, and with ulterior motives to learn about electronic computation, Mauchly volunteered for a crash course in electrical engineering at the University of Pennsylvania's Moore School of Engineering. Upon completion of this program, Mauchly accepted a teaching position at the Moore School, where he taught a brilliant young student, J. Presper Eckert. Mauchly and Eckert found a mutual interest in building an electronic calculating device. In order to secure the funding they needed to build their machine, they wrote a formal proposal for review by the school. They portrayed their machine as conservatively as they could, billing it as an "automatic calculator." Although they probably knew that computers would be able to function most efficiently using the binary numbering system, Mauchly and Eckert



HOW MUCH IS $\sqrt[3]{2589^{16}}$?

The Army's ENIAC can give you the answer in a fraction of a second!

Think that's a stumper? You should see *some* of the ENIAC's problems! Brain twisters that if put to paper would run off this page and feet beyond . . . addition, subtraction, multiplication, division—square root, cube root, any root. Solved by an incredibly complex system of circuits operating 18,000 electronic tubes and tipping the scales at 30 tons!

The ENIAC is symbolic of many amazing Army devices with a brilliant future for you! The new Regular Army needs men with aptitude for scientific work, and as one of the first trained in the post-war era, you stand to get in on the ground floor of important jobs

**YOUR REGULAR ARMY SERVES THE NATION
AND MANKIND IN WAR AND PEACE**

which have never before existed. You'll find that an Army career pays off.

The most attractive fields are filling quickly. Get into the swim while the getting's good! 1½, 2 and 3 year enlistments are open in the Regular Army to ambitious young men 18 to 34 (17 with parents' consent) who are otherwise qualified. If you enlist for 3 years, you may choose your own branch of the service, of those still open. Get full details at your nearest Army Recruiting Station.

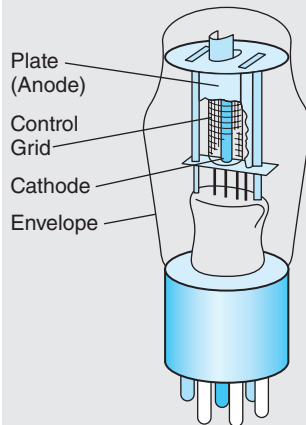
A GOOD JOB FOR YOU
U. S. Army
CHOOSE THIS
FINE PROFESSION NOW!

U.S. Army, 1946.

designed their system to use base 10 numbers, in keeping with the appearance of building a huge electronic adding machine. The university rejected Mauchly and Eckert's proposal. Fortunately, the U.S. Army was more interested.

During World War II, the army had an insatiable need for calculating the trajectories of its new ballistic armaments. Thousands of human "computers" were engaged around the clock cranking through the arithmetic required for these firing tables. Realizing that an electronic device could shorten ballistic table calculation from days to minutes, the army funded the ENIAC. And the ENIAC did indeed shorten the time to calculate a table from 20 hours to 30 seconds. Unfortunately, the machine wasn't ready before the end of the war. But the ENIAC had shown that vacuum tube computers were fast and feasible. During the next decade, vacuum tube systems continued to improve and were commercially successful.

What Is a Vacuum Tube?

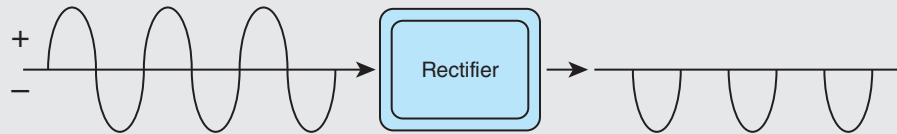


The wired world that we know today was born from the invention of a single electronic device called a **vacuum tube** by Americans and—more accurately—a **valve** by the British. Vacuum tubes should be called valves because they control the flow of electrons in electrical systems in much the same way as valves control the flow of water in a plumbing system. In fact, some mid-twentieth-century breeds of these electron tubes contain no vacuum at all, but are filled with conductive gases, such as mercury vapor, which can provide desirable electrical behavior.

The electrical phenomenon that makes tubes work was discovered by Thomas A. Edison in 1883 while he was trying to find ways to keep the filaments of his light bulbs from burning away (or oxidizing) a few minutes after electrical current was applied. Edison reasoned correctly that one way to prevent filament oxidation would be to place the filament in a vacuum. Edison didn't immediately understand that air not only supports combustion, but also is a good insulator. When he energized the electrodes holding a new tungsten filament, the filament soon became hot and burned out as the others had before it. This time, however, Edison noticed that electricity continued to flow from the warmed negative terminal to the cool positive terminal within the light bulb. In 1911, Owen Willans Richardson analyzed this behavior. He concluded that when a negatively charged filament was heated, electrons "boiled off" as water molecules can be boiled to create steam. He aptly named this phenomenon **thermionic emission**.

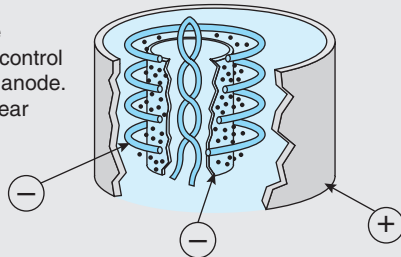
Thermionic emission, as Edison had documented it, was thought by many to be only an electrical curiosity. But in 1905, a British former assistant to Edison, John A. Fleming, saw Edison's discovery as much more than a novelty. He knew that thermionic emission supported the flow of electrons in only one direction: from the negatively charged **cathode** to the positively charged **anode**, also called a **plate**. He realized that this behavior could *rectify* alternating current. That is, it could change

alternating current into the direct current that was essential for the proper operation of telegraph equipment. Fleming used his ideas to invent an electronic valve later called a **diode tube** or **rectifier**.

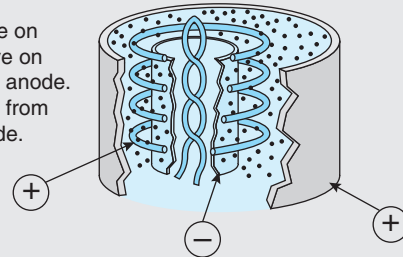


The diode was well suited for changing alternating current into direct current, but the greatest power of the electron tube was yet to be discovered. In 1907, an American named Lee DeForest added a third element, called a **control grid**. The control grid, when carrying a negative charge, can reduce or prevent electron flow from the cathode to the anode of a diode.

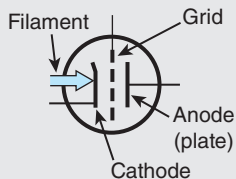
Negative charge on cathode and control grid; positive on anode. Electrons stay near cathode.



Negative charge on cathode; positive on control grid and anode. Electrons travel from cathode to anode.



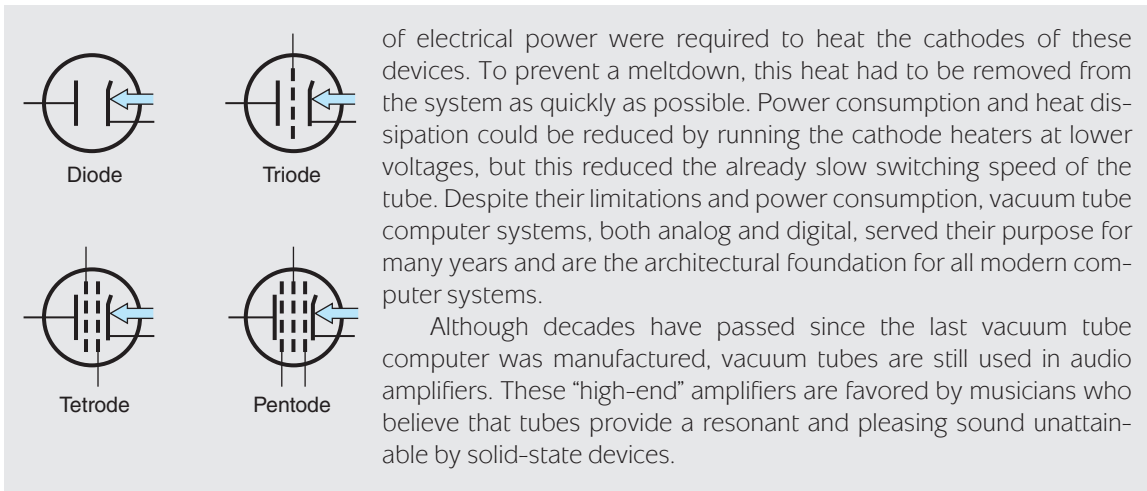
When DeForest patented his device, he called it an **audion tube**. It was later known as a **triode**. The schematic symbol for the triode is shown at the left.



A triode can act as either a switch or an amplifier. Small changes in the charge of the control grid can cause much larger changes in the flow of electrons between the cathode and the anode. Therefore, a weak signal applied to the grid results in a much stronger signal at the plate output. A sufficiently large negative charge applied to the grid stops all electrons from leaving the cathode.

Additional control grids were eventually added to the triode to allow more exact control of the electron flow. Tubes with two grids (four elements) are called **tetrodes**; tubes with three grids are called **pentodes**. Triodes and pentodes were the tubes most commonly used in communications and computer applications. Often, two or three triodes or pentodes would be combined within one envelope so they could share a single heater, thereby reducing the power consumption of a particular device. These latter-day devices were called “miniature” tubes because many were about 2 inches (5cm) high and 0.5 inch (1.5cm) in diameter. Equivalent full-sized diodes, triodes, and pentodes were a little smaller than a household light bulb.

Vacuum tubes were not well suited for building computers. Even the simplest vacuum tube computer system required thousands of tubes. Enormous amounts

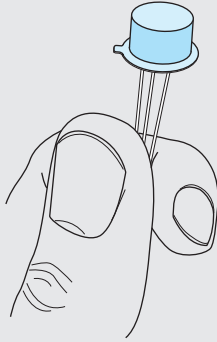


1.5.3 The Second Generation: Transistorized Computers (1954–1965)

The vacuum tube technology of the first generation was not very dependable. In fact, some ENIAC detractors believed that the system would never run because the tubes would burn out faster than they could be replaced. Although system reliability wasn’t as bad as the doomsayers predicted, vacuum tube systems often experienced more downtime than uptime.

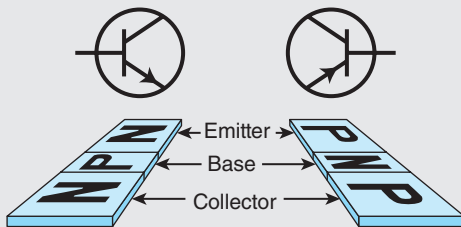
In 1948, three researchers with Bell Laboratories—John Bardeen, Walter Brattain, and William Shockley—invented the transistor. This new technology not only revolutionized devices such as televisions and radios, but also pushed the computer industry into a new generation. Because transistors consume less power than vacuum tubes, are smaller, and work more reliably, the circuitry in computers consequently became smaller and more reliable. Despite using transistors, computers of this generation were still bulky and quite costly. Typically only universities, governments, and large businesses could justify the expense. Nevertheless, a plethora of computer makers emerged in this generation; IBM, Digital Equipment Corporation (DEC), and Univac (now Unisys) dominated the industry. IBM marketed the 7094 for scientific applications and the 1401 for business applications. DEC was busy manufacturing the PDP-1. A company founded (but soon sold) by Mauchly and Eckert built the Univac systems. The most successful Unisys systems of this generation belonged to its 1100 series. Another company, Control Data Corporation (CDC), under the supervision of Seymour Cray, built the CDC 6600, the world’s first supercomputer. The \$10 million CDC 6600 could perform 10 million instructions per second, used 60-bit words, and had an astounding 128 kilowords of main memory.

What Is a Transistor?



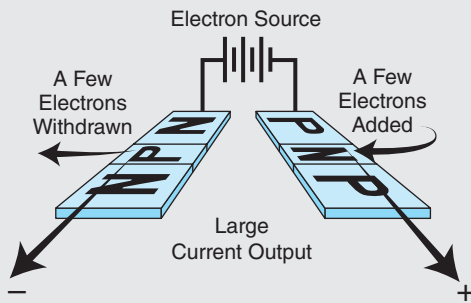
The transistor, short for **transfer resistor**, is the solid-state version of the triode. There is no such thing as a solid-state version of the tetrode or pentode. Electrons are better behaved in a solid medium than in the open void of a vacuum tube, so there is no need for the extra controlling grids. Either germanium or silicon can be the basic "solid" used in these solid-state devices. In their pure form, neither of these elements is a good conductor of electricity. But when they are combined with trace amounts of elements that are their neighbors in the Periodic Chart of the Elements, they conduct electricity in an effective and easily controlled manner.

Boron, aluminum, and gallium can be found to the left of silicon and germanium on the Periodic Chart. Because they lie to the left of silicon and germanium, they have one less electron in their outer electron shell, or **valence**. So if you add a small amount of aluminum to silicon, the silicon ends up with a slight imbalance in its outer electron shell, and therefore attracts electrons from any pole that has a negative potential (an excess of electrons). When modified (or **doped**) in this way, silicon or germanium becomes a **P-type** material.



Similarly, if we add a little boron, arsenic, or gallium to silicon, we'll have extra electrons in valences of the silicon crystals. This gives us an **N-type** material. A small amount of current will flow through the N-type material if we provide the loosely bound electrons in the N-type material with a place to go. In other words, if we apply a positive potential to N-type material, electrons will flow from the negative pole to the positive pole. If the poles are reversed, that is, if we apply a negative potential to the N-type material and a positive potential to the P-type material, no current will flow. This means we can make a solid-state diode from a simple junction of N- and P-type materials.

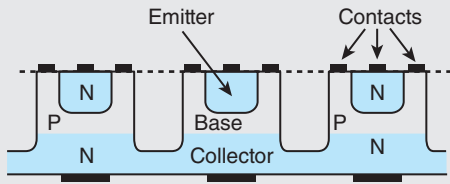
The solid-state triode, the transistor, consists of three layers of semiconductor material. Either a slice of P-type material is sandwiched between two N-type materials, or a slice of N-type material is sandwiched between two P-type materials. The former is called an NPN transistor, the latter a PNP transistor. The inner layer of the transistor is called the base; the other two layers are called the collector and the emitter.



The figure at the left shows how current flows through NPN and PNP transistors. The base in a transistor works just like the control grid in a triode tube: Small changes in the current at the base of a transistor result in a large electron flow from the emitter to the collector.

A **discrete-component** transistor is shown in "TO-50" packaging in the figure at the top of this sidebar. There are only three wires (leads) that connect the base, emitter, and collector of the transistor to the rest of the circuit. Transistors are not only smaller than vacuum tubes, but they also run cooler and are much

more reliable. Vacuum tube filaments, like light bulb filaments, run hot and eventually burn out. Computers using transistorized components will naturally be smaller and run cooler than their vacuum tube predecessors. The ultimate miniaturization, however, is not realized by replacing individual triodes with discrete transistors, but in shrinking entire circuits onto one piece of silicon.



Integrated circuits, or **chips**, contain hundreds to billions of microscopic transistors. Several different techniques are used to manufacture integrated circuits. One of the simplest methods involves creating a circuit using computer-aided design software that can print large maps of each of the several silicon layers forming the chip. Each map is used like a photographic negative where light-induced changes in a photoresistive substance on

the chip's surface produce the delicate patterns of the circuit when the silicon chip is immersed in a chemical that washes away the exposed areas of the silicon. This technique is called **photomicrolithography**. After the etching is completed, a layer of N-type or P-type material is deposited on the bumpy surface of the chip. This layer is then treated with a photoresistive substance, exposed to light, and etched as was the layer before it. This process continues until all the layers have been etched. The resulting peaks and valleys of P- and N-type material form microscopic electronic components, including transistors, that behave just like larger versions fashioned from discrete components, except that they run a lot faster and consume a small fraction of the power.

1.5.4 The Third Generation: Integrated Circuit Computers (1965–1980)

The real explosion in computer use came with the integrated circuit generation. Jack Kilby invented the integrated circuit (IC), or **microchip**, made of germanium. Six months later, Robert Noyce (who had also been working on integrated circuit design) created a similar device using silicon instead of germanium. This is the silicon chip upon which the computer industry was built. Early ICs allowed dozens of transistors to exist on a single silicon chip that was smaller than a single “discrete component” transistor. Computers became faster, smaller, and cheaper, bringing huge gains in processing power. The IBM System/360 family of computers was among the first commercially available systems to be built entirely of solid-state components. The 360 product line was also IBM's first offering in which all the machines in the family were compatible, meaning they all used the same assembly language. Users of smaller machines could upgrade to larger systems without rewriting all their software. This was a revolutionary new concept at the time.

The IC generation also saw the introduction of time-sharing and multiprogramming (the ability for more than one person to use the computer at a time). Multiprogramming, in turn, necessitated the introduction of new operating systems for these computers. Time-sharing minicomputers such as DEC's PDP-8 and PDP-11 made computing affordable to smaller businesses and more universities.

IC technology also allowed for the development of more powerful supercomputers. Seymour Cray took what he had learned while building the CDC 6600 and started his own company, the Cray Research Corporation. This company produced a number of supercomputers, starting with the \$8.8 million Cray-1, in 1976. The Cray-1, in stark contrast to the CDC 6600, could execute more than 160 million instructions per second and could support 8MB of memory. See Figure 1.2 for a size comparison of vacuum tubes, transistors, and integrated circuits.

1.5.5 The Fourth Generation: VLSI Computers (1980-????)

In the third generation of electronic evolution, multiple transistors were integrated onto one chip. As manufacturing techniques and chip technologies advanced, increasing numbers of transistors were packed onto one chip. There are now various levels of integration: SSI (small-scale integration), in which there are 10 to 100 components per chip; MSI (medium-scale integration), in which there are 100 to 1000 components per chip; LSI (large-scale integration), in which there are

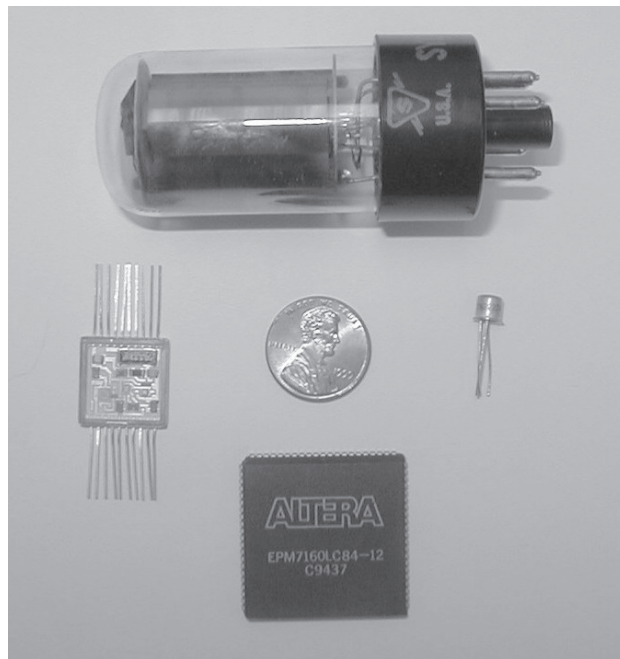


FIGURE 1.2 Comparison of Computer Components

Clockwise, starting from the top:

- 1) Vacuum tube
- 2) Transistor
- 3) Chip containing 3200 2-input NAND gates
- 4) Integrated circuit package (the small silver square in the lower left-hand corner is an integrated circuit)

Courtesy of Linda Null.

1000 to 10,000 components per chip; and finally, VLSI (very-large-scale integration), in which there are more than 10,000 components per chip. This last level, VLSI, marks the beginning of the fourth generation of computers. The complexity of integrated circuits continues to grow, with more transistors being added all the time. The term *ULSI* (ultra-large-scale integration) has been suggested for integrated circuits containing more than 1 million transistors. In 2005, billions of transistors were put on a single chip. Other useful terminology includes: (1) WSI (wafer-scale integration, building superchip ICs from an entire silicon wafer; (2) 3D-IC (three-dimensional integrated circuit); and (3) SOC (system-on-a-chip), an IC that includes all the necessary components for the entire computer.

To give some perspective to these numbers, consider the ENIAC-on-a-chip project. In 1997, to commemorate the fiftieth anniversary of its first public demonstration, a group of students at the University of Pennsylvania constructed a single-chip equivalent of the ENIAC. The 1800-square-foot, 30-ton beast that devoured 174 kilowatts of power the minute it was turned on had been reproduced on a chip the size of a thumbnail. This chip contained approximately 174,569 transistors—an order of magnitude fewer than the number of components typically placed on the same amount of silicon in the late 1990s.

VLSI allowed Intel, in 1971, to create the world's first microprocessor, the 4004, which was a fully functional, 4-bit system that ran at 108KHz. Intel also introduced the random access memory (RAM) chip, accommodating four kilobits of memory on a single chip. This allowed computers of the fourth generation to become smaller and faster than their solid-state predecessors.

VLSI technology, and its incredible shrinking circuits, spawned the development of microcomputers. These systems were small enough and inexpensive enough to make computers available and affordable to the general public. The premiere microcomputer was the Altair 8800, released in 1975 by the Micro Instrumentation and Telemetry (MITS) corporation. The Altair 8800 was soon followed by the Apple I and Apple II, and Commodore's PET and Vic 20. Finally, in 1981, IBM introduced its PC (Personal Computer).

The Personal Computer was IBM's third attempt at producing an "entry-level" computer system. Its Datamaster and its 5100 Series desktop computers flopped miserably in the marketplace. Despite these early failures, IBM's John Opel convinced his management to try again. He suggested forming a fairly autonomous "independent business unit" in Boca Raton, Florida, far from IBM's headquarters in Armonk, New York. Opel picked Don Estridge, an energetic and capable engineer, to champion the development of the new system, code-named the Acorn. In light of IBM's past failures in the small-systems area, corporate management held tight rein on the Acorn's timeline and finances. Opel could get his project off the ground only after promising to deliver it within a year, a seemingly impossible feat.

Estridge knew that the only way he could deliver the PC within the wildly optimistic 12-month schedule would be to break with IBM convention and use as many "off-the-shelf" parts as possible. Thus, from the outset, the IBM PC was conceived with an "open" architecture. Although some people at IBM may have later regretted the decision to keep the architecture of the PC as nonproprietary

as possible, it was this very openness that allowed IBM to set the standard for the industry. While IBM's competitors were busy suing companies for copying their system designs, PC clones proliferated. Before long, the price of "IBM-compatible" microcomputers came within reach for just about every small business. Also, thanks to the clone makers, large numbers of these systems soon began finding true "personal use" in people's homes.

IBM eventually lost its microcomputer market dominance, but the genie was out of the bottle. For better or worse, the IBM architecture continues to be the de facto standard for microcomputing, with each year heralding bigger and faster systems. Today, the average desktop computer has many times the computational power of the mainframes of the 1960s.

Since the 1960s, mainframe computers have seen stunning improvements in price-performance ratios owing to VLSI technology. Although the IBM System/360 was an entirely solid-state system, it was still a water-cooled, power-gobbling behemoth. It could perform only about 50,000 instructions per second and supported only 16MB of memory (while usually having *kilobytes* of physical memory installed). These systems were so costly that only the largest businesses and universities could afford to own or lease one. Today's mainframes—now called "enterprise servers"—are still priced in the millions of dollars, but their processing capabilities have grown several thousand times over, passing the billion-instructions-per-second mark in the late 1990s. These systems, often used as Web servers, routinely support hundreds of thousands of transactions *per minute*!

The processing power brought by VLSI to supercomputers defies comprehension. The first supercomputer, the CDC 6600, could perform 10 million instructions per second, and had 128KB of main memory. By contrast, supercomputers of today contain thousands of processors, can address terabytes of memory, and will soon be able to perform a *quadrillion* instructions per second.

What technology will mark the beginning of the fifth generation? Some say the fifth generation will mark the acceptance of parallel processing and the use of networks and single-user workstations. Many people believe we have already crossed into this generation. Some believe it will be quantum computing. Some people characterize the fifth generation as being the generation of neural network, DNA, or optical computing systems. It's possible that we won't be able to define the fifth generation until we have advanced into the sixth or seventh generations, and whatever those eras will bring.

The Integrated Circuit and Its Production

Integrated circuits are found all around us, from computers to cars to refrigerators to cell phones. The most advanced circuits contain hundreds of millions (and even billions) of components in an area about the size of your thumbnail. The transistors in these advanced circuits can be as small as 45nm, or 0.000045 millimeters, in size. Thousands of these transistors would fit in a circle the diameter of a human hair.

How are these circuits made? They are manufactured in semiconductor fabrication facilities. Because the components are so small, all precautions must be taken to ensure a sterile, particle-free environment, so manufacturing is done in a “clean room.” There can be no dust, no skin cells, no smoke—not even bacteria. Workers must wear clean room suits, often called “bunny suits,” to ensure that even the tiniest particle does not escape into the air.

The process begins with the chip design, which eventually results in a mask, the template or blueprint that contains the circuit patterns. A silicon wafer is then covered by an insulating layer of oxide, followed by a layer of photosensitive film called photo-resist. This photo-resist has regions that break down under UV light and other regions that do not. A UV light is then shone through the mask (a process called photolithography). Bare oxide is left on portions where the photo-resist breaks down under the UV light. Chemical “etching” is then used to dissolve the revealed oxide layer and also to remove the remaining photo-resist not affected by the UV light. The “doping” process embeds certain impurities into the silicon that alters the electrical properties of the unprotected areas, basically creating the transistors. The chip is then covered with another layer of both the insulating oxide material and the photo-resist, and the entire process is repeated hundreds of times, each iteration creating a new layer of the chip. Different masks are used with a similar process to create the wires that connect the components on the chip. The circuit is finally encased in a protective plastic cover, tested, and shipped out.

As components become smaller and smaller, the equipment used to make them must be of continually higher quality. This has resulted in a dramatic increase in the cost of manufacturing ICs over the years. In the early 1980s, the cost to build a semiconductor factory was roughly \$10 million. By the late 1980s, that cost had risen to approximately \$200 million, and by the late 1990s, an IC fabrication factory cost more or less around \$1 billion. In 2005, Intel spent approximately \$2 billion for a single fabrication facility and, in 2007, invested roughly \$7 billion to retool three plants in order to allow them to produce a smaller processor. In 2009, AMD began building a \$4.2 billion chip manufacturing facility in upstate New York.

The manufacturing facility is not the only high-dollar item when it comes to making ICs. The cost to design a chip and create the mask can run anywhere from \$1 million to \$3 million—more for smaller chips and less for larger ones. Considering the costs of both the chip design and the fabrication facility, it truly is amazing that we can walk into our local computer store and buy a new Intel i3 microprocessor chip for around \$100.

1.5.6 Moore’s Law

So where does it end? How small can we make transistors? How densely can we pack chips? No one can say for sure. Every year, scientists continue to thwart prognosticators’ attempts to define the limits of integration. In fact, more than one skeptic raised an eyebrow when, in 1965, Intel founder Gordon Moore stated, “The density of transistors in an integrated circuit will double every year.” The current version of this prediction is usually conveyed as “the density of silicon chips doubles every 18 months.” This assertion has become

known as **Moore's Law**. Moore intended this postulate to hold for only 10 years. However, advances in chip manufacturing processes have allowed this assertion to hold for almost 40 years (and many believe it will continue to hold well into the 2010s).

Yet, using current technology, Moore's Law cannot hold forever. There are physical and financial limitations that must ultimately come into play. At the current rate of miniaturization, it would take about 500 years to put the entire solar system on a chip! Clearly, the limit lies somewhere between here and there. Cost may be the ultimate constraint. **Rock's Law**, proposed by early Intel capitalist Arthur Rock, is a corollary to Moore's Law: "The cost of capital equipment to build semiconductors will double every four years." Rock's Law arises from the observations of a financier who saw the price tag of new chip facilities escalate from about \$12,000 in 1968 to \$12 million in the mid-1990s. In 2005, the cost of building a new chip plant was nearing \$3 billion. At this rate, by the year 2035, not only will the size of a memory element be smaller than an atom, but it would also require the entire wealth of the world to build a single chip! So even if we continue to make chips smaller and faster, the ultimate question may be whether we can afford to build them.

Certainly, if Moore's Law is to hold, Rock's Law must fall. It is evident that for these two things to happen, computers must shift to a radically different technology. Research into new computing paradigms has been proceeding in earnest during the last half decade. Laboratory prototypes fashioned around organic computing, superconducting, molecular physics, and quantum computing have been demonstrated. Quantum computers, which leverage the vagaries of quantum mechanics to solve computational problems, are particularly exciting. Not only would quantum systems compute exponentially faster than any previously used method, but they would also revolutionize the way in which we define computational problems. Problems that today are considered ludicrously infeasible could be well within the grasp of the next generation's schoolchildren. These schoolchildren may, in fact, chuckle at our "primitive" systems in the same way that we are tempted to chuckle at the ENIAC.

1.6 THE COMPUTER LEVEL HIERARCHY

If a machine is to be capable of solving a wide range of problems, it must be able to execute programs written in different languages, from Fortran and C to Lisp and Prolog. As we shall see in Chapter 3, the only physical components we have to work with are wires and gates. A formidable open space—a **semantic gap**—exists between these physical components and a high-level language such as C++. For a system to be practical, the semantic gap must be invisible to most of the users of the system.

Programming experience teaches us that when a problem is large, we should break it down and use a "divide and conquer" approach. In programming, we divide a problem into modules and then design each module separately. Each module performs a specific task, and modules need only know how to interface with other modules to make use of them.

Computer system organization can be approached in a similar manner. Through the principle of abstraction, we can imagine the machine to be built from a hierarchy of levels, in which each level has a specific function and exists as a distinct hypothetical machine. We call the hypothetical computer at each level a **virtual machine**. Each level's virtual machine executes its own particular set of instructions, calling upon machines at lower levels to carry out the tasks when necessary. By studying computer organization, you will see the rationale behind the hierarchy's partitioning, as well as how these layers are implemented and interface with each other. Figure 1.3 shows the commonly accepted layers representing the abstract virtual machines.

Level 6, the User Level, is composed of applications and is the level with which everyone is most familiar. At this level, we run programs such as word processors, graphics packages, or games. The lower levels are nearly invisible from the User Level.

Level 5, the High-Level Language Level, consists of languages such as C, C++, Fortran, Lisp, Pascal, and Prolog. These languages must be translated

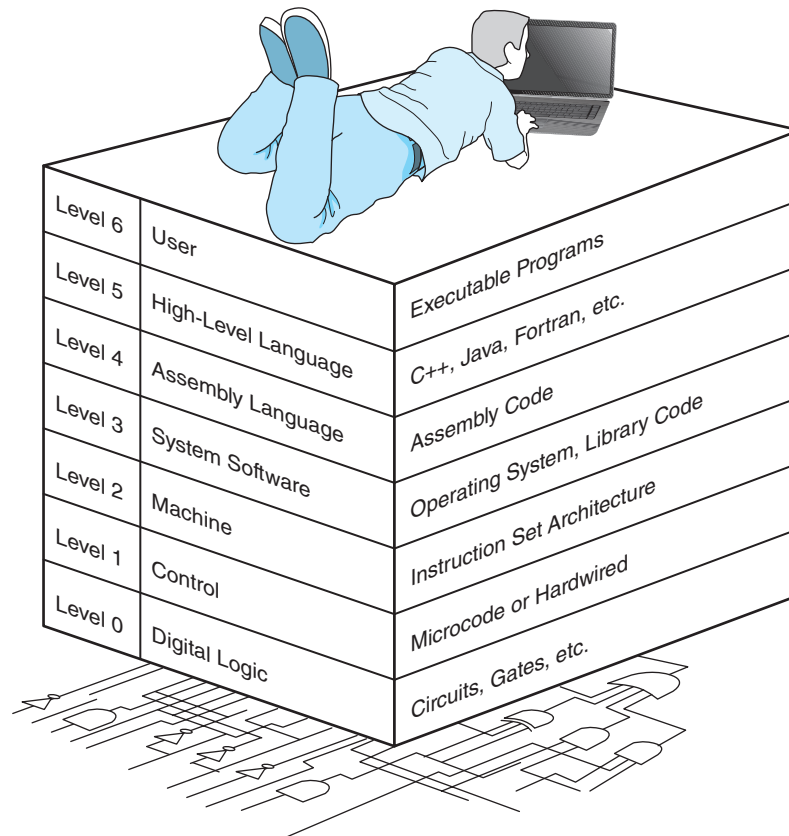


FIGURE 1.3 The Abstract Levels of Modern Computing Systems

(using either a compiler or an interpreter) to a language the machine can understand. Compiled languages are translated into assembly language and then assembled into machine code. (They are translated to the next lower level.) The user at this level sees very little of the lower levels. Even though a programmer must know about data types and the instructions available for those types, he or she need not know about how those types are actually implemented.

Level 4, the Assembly Language Level, encompasses some type of assembly language. As previously mentioned, compiled higher-level languages are first translated to assembly, which is then directly translated to machine language. This is a one-to-one translation, meaning that one assembly language instruction is translated to exactly one machine language instruction. By having separate levels, we reduce the semantic gap between a high-level language, such as C++, and the actual machine language (which consists of 0s and 1s).

Level 3, the System Software Level, deals with operating system instructions. This level is responsible for multiprogramming, protecting memory, synchronizing processes, and various other important functions. Often, instructions translated from assembly language to machine language are passed through this level unmodified.

Level 2, the Instruction Set Architecture (ISA), or Machine Level, consists of the machine language recognized by the particular architecture of the computer system. Programs written in a computer's true machine language on a hardwired computer (see below) can be executed directly by the electronic circuits without any interpreters, translators, or compilers. We will study ISAs in depth in Chapters 4 and 5.

Level 1, the Control Level, is where a **control unit** makes sure that instructions are decoded and executed properly and that data is moved where and when it should be. The control unit interprets the machine instructions passed to it, one at a time, from the level above, causing the required actions to take place.

Control units can be designed in one of two ways: They can be **hardwired** or they can be **microprogrammed**. In hardwired control units, control signals emanate from blocks of digital logic components. These signals direct all the data and instruction traffic to appropriate parts of the system. Hardwired control units are typically very fast because they are actually physical components. However, once implemented, they are very difficult to modify for the same reason.

The other option for control is to implement instructions using a microprogram. A microprogram is a program written in a low-level language that is implemented directly by the hardware. Machine instructions produced in Level 2 are fed into this microprogram, which then interprets the instructions by activating hardware suited to execute the original instruction. One machine-level instruction is often translated into several microcode instructions. This is not the one-to-one correlation that exists between assembly language and machine language. Microprograms are popular because they can be modified relatively easily. The disadvantage of microprogramming is, of course, that the additional layer of translation typically results in slower instruction execution.

Level 0, the Digital Logic Level, is where we find the physical components of the computer system: the gates and wires. These are the fundamental building

blocks, the implementations of the mathematical logic, that are common to all computer systems. Chapter 3 presents the Digital Logic Level in detail.

1.7 CLOUD COMPUTING: COMPUTING AS A SERVICE

We must never forget that the ultimate aim of every computer system is to deliver functionality to its users. Computer users typically do not care about terabytes of storage and gigahertz of processor speed. In fact, many companies and government agencies have “gotten out of the technology business” entirely by outsourcing their data centers to third-party specialists. These outsourcing agreements tend to be highly complex and prescribe every aspect of the hardware configuration. Along with the detailed hardware specifications, **service-level agreements (SLAs)** provide penalties if certain parameters of system performance and availability are not met. Both contracting parties employ individuals whose main job is to monitor the contract, calculate bills, and determine SLA penalties when needed. Thus, with the additional administrative overhead, data center outsourcing is neither a cheap nor an easy solution for companies that want to avoid the problems of technology management.

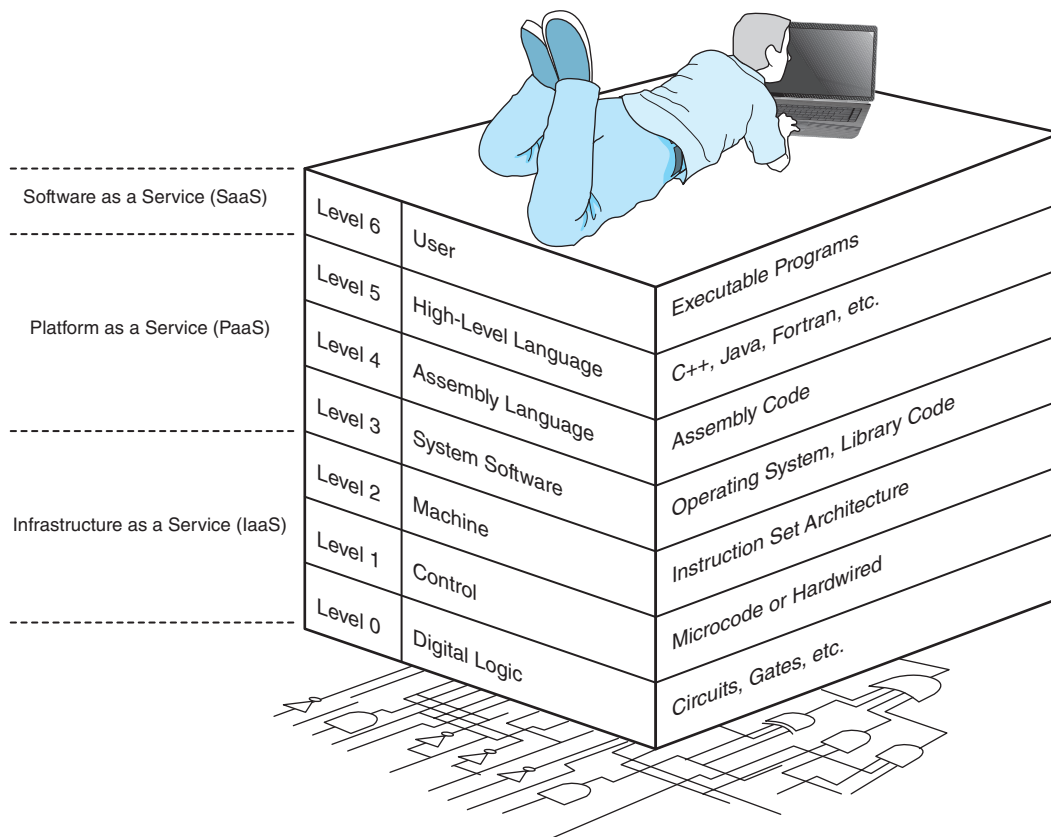


FIGURE 1.4 Levels of Computing as a Service

A somewhat easier approach may be found in the emerging field of Cloud computing. **Cloud computing** is the general term for any type of virtual computing platform provided over the Internet. A Cloud computing platform is defined in terms of the services that it provides rather than its physical configuration. Its name derives from the cloud icon that symbolizes the Internet on schematic diagrams. But the metaphor carries well into the actual Cloud infrastructure, because the computer is more abstract than real. The “computer” and “storage” appear to the user as a single entity in the Cloud but usually span several physical servers. The storage is usually located on an array of disks that are not directly connected to any particular server. System software is designed to give this configuration the illusion of being a single system; thus, we say that it presents a **virtual machine** to the user.

Cloud computing services can be defined and delivered in a number of ways based on levels of the computer hierarchy shown again in Figure 1.4. At the top of the hierarchy, where we have executable programs, a Cloud provider might offer an entire application over the Internet, with no components installed locally. This is called **Software as a Service**, or **SaaS**. The consumer of this service does not maintain the application or need to be at all concerned with the infrastructure in any way. SaaS applications tend to focus on narrow, non-business-critical applications. Well-known examples include Gmail, Dropbox, GoToMeeting, and Netflix. Specialized products are available for tax return preparation, payroll, fleet management, and case management, to name only a few. Salesforce.com is a pioneering, full-featured SaaS offering designed for customer relationship management. Fee-based SaaS is typically billed monthly according to the number of users, sometimes with per-transaction fees added on as well.

A great disadvantage of SaaS is that the consumer has little control over the behavior of the product. This may be problematic if a company has to make radical changes to its processes or policies in order to use a SaaS product. Companies that desire to have more control over their applications, or that need applications for which SaaS is unavailable, might instead opt to deploy their own applications on a Cloud-hosted environment called **Platform as a Service**, or **PaaS**. PaaS provides server hardware, operating systems, database services, security components, and backup and recovery services. The PaaS provider manages performance and availability of the environment, whereas the customer manages the applications hosted in the PaaS Cloud. The customer is typically billed monthly per megabytes of storage, processor utilization, and megabytes of data transferred. Well-known PaaS providers include Google App Engine and Microsoft Windows Azure Cloud Services [as well as Force.com (PaaS provided by Salesforce.com)].

PaaS is not a good fit in situations where rapid configuration changes are required. This would be the case if a company’s main business is software development. The formality of change processes necessary to a well-run PaaS operation impedes rapid software deployment [by forcing a company to play by the service provider’s rules]. Indeed, in any company where staff is capable of managing operating system and database software, the **Infrastructure as a Service (IaaS)** Cloud model might be the best option. IaaS, [the most basic of the models,]

provides only server hardware, secure network access to the servers, and backup and recovery services. The customer is responsible for all system software including the operating system and databases. IaaS is typically billed by the number of virtual machines used, megabytes of storage, and megabytes of data transferred, but at a lower rate than PaaS. The biggest names in IaaS include Amazon EC2, Google Compute Engine, Microsoft Azure Services Platform, Rackspace, and HP Cloud.

Not only do PaaS and IaaS liberate the customer from the difficulties of data center management, they also provide **elasticity**: the ability to add and remove resources based on demand. A customer pays for only as much infrastructure as is needed. So if a business has a peak season, extra capacity needs to be allocated only for the duration of the peak period. This flexibility can save a company a great deal of money when it has large variations in computing demands.

Cloud storage is a limited type of IaaS. The general public can obtain small amounts of Cloud storage inexpensively through services such as Dropbox, Google Drive, and Amazon.com's Cloud Drive—to name only a few among a crowded field. Google, Amazon, HP, IBM, and Microsoft are among several vendors that provide Cloud storage for the enterprise. As with Cloud computing in general, enterprise-grade Cloud storage also requires careful management of performance and availability.

The question that all potential Cloud computing customers must ask themselves is whether it is less expensive to maintain their own data center or to buy Cloud services—including the allowances for peak periods. Moreover, as with traditional outsourcing, vendor-provided Cloud computing still involves considerable contract negotiation and management on the part of both parties. SLA management remains an important activity in the relationship between the service provider and the service consumer. Moreover, once an enterprise moves its assets to the Cloud, it might be difficult to transition back to a company-owned data center, should the need arise. Thus, any notion of moving assets to the Cloud must be carefully considered, and the risks clearly understood.

The Cloud also presents a number of challenges to computer scientists. First and foremost is the technical configuration of the data center. The infrastructure must provide for uninterrupted service, even during maintenance activities. It must permit expedient allocation of capacity to where it is needed without degrading or interrupting services. Performance of the infrastructure must be carefully monitored and interventions taken whenever performance falls below certain defined thresholds; otherwise, monetary SLA penalties may be incurred.

On the consumer side of the Cloud, software architects and programmers must be mindful of resource consumption, because the Cloud model charges fees in proportion to the resources consumed. These resources include communications bandwidth, processor cycles, and storage. Thus, to save money, application programs should be designed to reduce trips over the network, economize machine cycles, and minimize bytes of storage. Meticulous testing is crucial prior to deploying a program in the Cloud: An errant module that consumes resources, say, in an infinite loop, could result in a “surprising” Cloud bill at the end of the month.

With the cost and complexity of data centers continuing to rise—with no end in sight—Cloud computing is almost certain to become the platform of choice for medium- to small-sized businesses. But the Cloud is not worry-free. A company might end up trading its technical challenges for even more vexing supplier management challenges.

1.8 THE VON NEUMANN MODEL

In the earliest electronic computing machines, programming was synonymous with connecting wires to plugs. No layered architecture existed, so programming a computer was as much of a feat of electrical engineering as it was an exercise in algorithm design. Before their work on the ENIAC was complete, John W. Mauchly and J. Presper Eckert conceived of an easier way to change the behavior of their calculating machine. They reckoned that memory devices, in the form of mercury delay lines, could provide a way to store program instructions. This would forever end the tedium of rewiring the system each time it had a new problem to solve, or an old one to debug. Mauchly and Eckert documented their idea, proposing it as the foundation for their next computer, the EDVAC. Unfortunately, while they were involved in the top secret ENIAC project during World War II, Mauchly and Eckert could not immediately publish their insight.

No such proscriptions, however, applied to a number of people working at the periphery of the ENIAC project. One of these people was a famous Hungarian mathematician named John von Neumann (pronounced *von noy-man*). After reading Mauchly and Eckert's proposal for the EDVAC, von Neumann published and publicized the idea. So effective was he in the delivery of this concept that history has credited him with its invention. All stored-program computers have come to be known as **von Neumann systems** using the **von Neumann architecture**. Although we are compelled by tradition to say that stored-program computers use the von Neumann architecture, we shall not do so without paying proper tribute to its true inventors: John W. Mauchly and J. Presper Eckert.

Today's version of the stored-program machine architecture satisfies at least the following characteristics:

- Consists of three hardware systems: A **central processing unit (CPU)** with a control unit, an **arithmetic logic unit (ALU)**, **registers** (small storage areas), and a program counter; a **main memory system**, which holds programs that control the computer's operation; and an **I/O system**.
- Capacity to carry out sequential instruction processing.
- Contains a single path, either physically or logically, between the main memory system and the control unit of the CPU, forcing alternation of instruction and execution cycles. This single path is often referred to as the **von Neumann bottleneck**.

Figure 1.5 shows how these features work together in modern computer systems. Notice that the system shown in the figure passes all of its I/O through

the arithmetic logic unit (actually, it passes through the accumulator, which is part of the ALU). This architecture runs programs in what is known as the **von Neumann execution cycle** (also called the **fetch-decode-execute cycle**), which describes how the machine works. One iteration of the cycle is as follows:

1. The control unit fetches the next program instruction from the memory, using the program counter to determine where the instruction is located.
2. The instruction is decoded into a language the ALU can understand.
3. Any data operands required to execute the instruction are fetched from memory and placed in registers in the CPU.
4. The ALU executes the instruction and places the results in registers or memory.

The ideas present in the von Neumann architecture have been extended so that programs and data stored in a slow-to-access storage medium, such as a hard disk, can be copied to a fast-access, volatile storage medium such as RAM prior to execution. This architecture has also been streamlined into what is currently called the **system bus model**, which is shown in Figure 1.6. The data bus moves data from main memory to the CPU registers (and vice versa). The address bus holds the

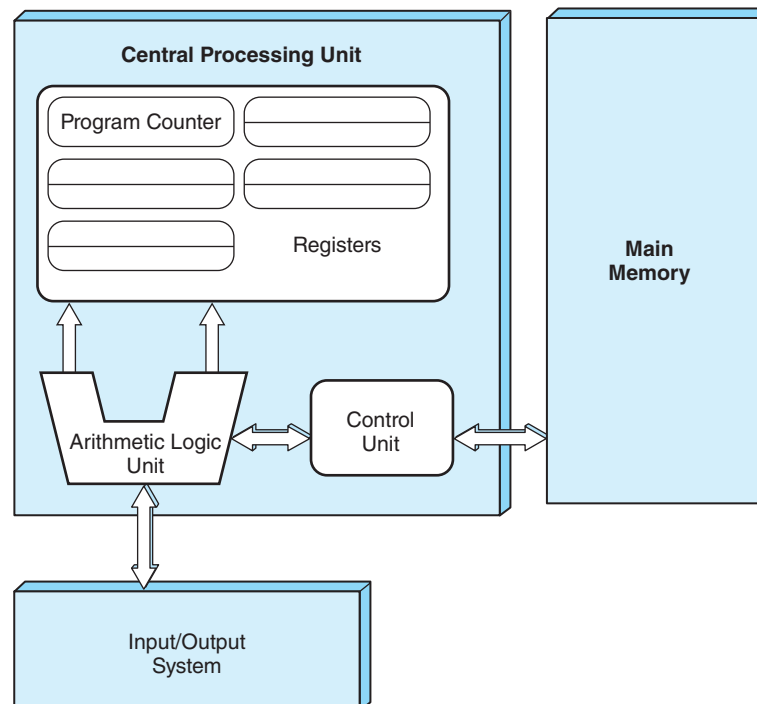


FIGURE 1.5 The von Neumann Architecture

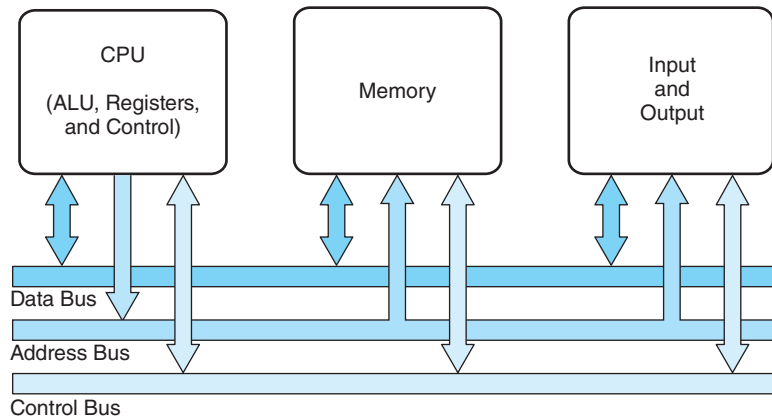


FIGURE 1.6 The Modified von Neumann Architecture, Adding a System Bus

address of the data that the data bus is currently accessing. The control bus carries the necessary control signals that specify how the information transfer is to take place.

Other enhancements to the von Neumann architecture include using index registers for addressing, adding floating-point data, using interrupts and asynchronous I/O, adding virtual memory, and adding general registers. You will learn a great deal about these enhancements in the chapters that follow.

Quantum Leap for Computers: How Small Can We Go?

VLSI technology has allowed us to put billions of transistors on a single chip, but there is a limit to how small we can go with current transistor technology. Researchers at the University of New South Wales' Centre for Quantum Computer Technology and the University of Wisconsin-Madison have taken "small" to an entirely new level. In May 2010, they announced the 7-atom transistor, a working transistor embedded in silicon that is only 7 atoms in size. Transistors 1 atom in size that allowed the flows of electrons were reported as early as 2002, but this transistor is different in that it provides all the functionality of a transistor as we know it today.

The 7-atom transistor was created by hand, using a scanning tunneling microscope. It's a long way from being mass produced, but the researchers hope to make it commercially available by 2015. The transistor's tiny size means smaller but more powerful computers. Experts estimate it may shrink microchips by a factor of 100, while enabling an exponential speedup in processing. This means our computers could become one hundred times smaller, but at the same time, also one hundred times faster.

In addition to replacing traditional transistors, this discovery may be fundamental in the efforts to build a quantum computer in silicon. Quantum computing is expected

to be the next significant leap in computer technology. Small quantum computers now exist that perform calculations millions of times faster than conventional computers, but these computers are too small to be of much use. A large-scale, working quantum computer would enable us to perform calculations and solve problems that would take a conventional computer more than 13 billion years. That could change the way we view the world. For one thing, every encryption algorithm employed today would be useless against that kind of computing power. On the other hand, ultra-secure communications would be possible using new quantum technologies.

Quantum computers have significant potential. Current applications, including special effects for movies, cryptography, searching large data files, factoring large numbers, simulating various systems (such as nuclear explosions and weather patterns), military and intelligence gathering, and intensive, time-consuming computations (such as those found in astronomy, physics, and chemistry), would all see tremendous performance increases if quantum computing were used. New applications we have not yet discovered are likely to evolve as well.

In addition to its potential to change computing as we know it today, this new 7-atom transistor is significant for another reason. Recall Moore's Law; this law is not so much a law of nature, but rather an expectation of innovation and a significant driving force in chip design. Moore's Law has held since 1965, but in order to do so, chip manufacturers have jumped from one technology to another. Gordon Moore himself has predicted that, if restricted to CMOS silicon, his law will fail sometime around 2020. The discovery of this 7-atom transistor gives new life to Moore's Law—and we suspect that Gordon Moore is breathing a sigh of relief over its discovery. However, noted physicist Stephen Hawking has explained that chip manufacturers are limited in their quest to “enforce” Moore's Law by two fundamental constraints: the speed of light and the atomic nature of matter, implying that Moore's Law will eventually fail, regardless of the technology being used.

1.9 NON-VON NEUMANN MODELS

Until recently, almost all general-purpose computers followed the von Neumann design. That is, the architecture consisted of a CPU, memory, and I/O devices, and they had single storage for instructions and data, as well as a single bus used for fetching instructions and transferring data. von Neumann computers execute instructions sequentially and are therefore extremely well suited to sequential processing. However, the von Neumann bottleneck continues to baffle engineers looking for ways to build fast systems that are inexpensive and compatible with the vast body of commercially available software.

Engineers who are not constrained by the need to maintain compatibility with von Neumann systems are free to use many different models of computing. Non-von Neumann architectures are those in which the model of computation varies from the characteristics listed for the von Neumann architecture. For example, an architecture that does not store programs and data in memory or does not process

a program sequentially would be considered a non-von Neumann machine. Also, a computer that has two buses, one for data and a separate one for instructions, would be considered a non-von Neumann machine. Computers designed using the **Harvard architecture** have two buses, thus allowing data and instructions to be transferred simultaneously, but also have separate storage for data and instructions. Many modern general-purpose computers use a modified version of the Harvard architecture in which they have separate pathways for data and instructions but not separate storage. Pure Harvard architectures are typically used in microcontrollers (an entire computer system on a chip), such as those found in embedded systems, as in appliances, toys, and cars.

Many non-von Neumann machines are designed for special purposes. The first recognized non-von Neumann processing chip was designed strictly for image processing. Another example is a **reduction machine** (built to perform combinatory logic calculations using graph reduction). Other non-von Neumann computers include **digital signal processors** (DSPs) and **media processors**, which can execute a single instruction on a set of data (instead of executing a single instruction on a single piece of data).

A number of different subfields fall into the non-von Neumann category, including **neural networks** (using ideas from models of the brain as a computing paradigm) implemented in silicon, **cellular automata**, **cognitive computers** (machines that learn by experience rather than through programming, including IBM's SyNAPSE computer, a machine that models the human brain), **quantum computation** (a combination of computing and quantum physics), **dataflow computation**, and **parallel computers**. These all have something in common—the computation is distributed among different processing units that act in parallel. They differ in how weakly or strongly the various components are connected. Of these, parallel computing is currently the most popular.

1.10 PARALLEL PROCESSORS AND PARALLEL COMPUTING

Today, parallel processing solves some of our biggest problems in much the same way that settlers of the Old West solved their biggest problems using parallel oxen. If they were using an ox to move a tree and the ox was not big enough or strong enough, they certainly didn't try to grow a bigger ox—they used two oxen. If our computer isn't fast enough or powerful enough, instead of trying to develop a faster, more powerful computer, why not simply use multiple computers? This is precisely what parallel computing does. The first parallel processing systems were built in the late 1960s and had only two processors. The 1970s saw the introduction of supercomputers with as many as 32 processors, and the 1980s brought the first systems with more than 1000 processors. Finally, in 1999, IBM announced funding for the development of a supercomputer architecture called the **Blue Gene** series. The first computer in this series, the **Blue Gene/L**, is a massively parallel computer containing 131,000 dual-core processors, each with its own dedicated memory. In addition to allowing researchers to study the behavior of protein folding (by using large simulations), this computer

has also allowed researchers to explore new ideas in parallel architectures and software for those architectures. IBM has continued to add computers to this series. The **Blue Gene/P** appeared in 2007 and has quad-core processors. The latest computer designed for this series, the **Blue Gene/Q**, uses 16-core processors, with 1024 compute nodes per rack, scalable up to 512 racks. Installations of the Blue Gene/Q computer include Nostromo (being used for biomedical data in Poland), Sequoia (being used at Lawrence Livermore National Laboratory for nuclear simulations and scientific research), and Mira (used at Argonne National Laboratory).

Dual-core and quad-core processors (and higher, as we saw in Blue Gene/Q) are examples of **multicore processors**. But what is a multicore processor? Essentially, it is a special type of parallel processor. Parallel processors are often classified as either “shared memory” processors (in which processors all share the same global memory) or “distributed memory” computers (in which each processor has its own private memory). Chapter 9 covers parallel processors in detail. The following discussion is limited to shared memory multicore architectures—the type used in personal computers.

Multicore architectures are parallel processing machines that allow for multiple processing units (often called **cores**) on a single chip. Dual core means 2 cores; quad core machines have 4 cores; and so on. But what is a core? Instead of a single processing unit in an integrated circuit (as found in typical von Neumann machines), independent multiple cores are “plugged in” and run in parallel. Each processing unit has its own ALU and set of registers, but all processors share memory and some other resources. “Dual core” is different from “dual processor.” Dual-processor machines, for example, have two processors, but each processor plugs into the motherboard separately. The important distinction to note is that all cores in multicore machines are integrated into the same chip. This means that you could, for example, replace a single-core (uniprocessor) chip in your computer with, for example, a dual-core processor chip (provided your computer had the appropriate socket for the new chip). Many computers today are advertised as dual core, quad core, or higher. Dual core is generally considered the standard in today’s computers. Although most desktop and laptop computers have limited cores (fewer than 8), machines with hundreds of cores are available for the right price, of course.

Just because your computer has multiple cores does not mean it will run your programs more quickly. Application programs (including operating systems) must be written to take advantage of multiple processing units (this statement is true for parallel processing in general). Multicore computers are very useful for **multitasking**—when users are doing more than one thing at a time. For example, you may be reading email, listening to music, browsing the Web, and burning a DVD all at the same time. These “multiple tasks” can be assigned to different processors and carried out in parallel, provided the operating system is able to manipulate many tasks at once.

In addition to multitasking, **multithreading** can also increase the performance of any application with inherent parallelism. Programs are divided into

threads, which can be thought of as mini-processes. For example, a Web browser is multithreaded; one thread can download text, while each image is controlled and downloaded by a separate thread. If an application is multithreaded, separate threads can run in parallel on different processing units. We should note that even on uniprocessors, multithreading can improve performance, but this is a discussion best left for another time. For more information, see Stallings (2012).

To summarize, parallel processing refers to a collection of different architectures, from multiple separate computers working together, to multiple processors sharing memory, to multiple cores integrated onto the same chip. Parallel processors are technically not classified as von Neumann machines because they do not process instructions sequentially. However, many argue that parallel processing computers contain CPUs, use program counters, and store both programs and data in main memory, which makes them more like an extension to the von Neumann architecture rather than a departure from it; these people view parallel processing computers as sets of cooperating von Neumann machines. In this regard, perhaps it is more appropriate to say that parallel processing exhibits “non–von Neumannness.” Regardless of how parallel processors are classified, parallel computing allows us to multitask and to solve larger and more complex problems, and is driving new research in various software tools and programming.

Even parallel computing has its limits, however. As the number of processors increases, so does the overhead of managing how tasks are distributed to those processors. Some parallel processing systems require extra processors just to manage the rest of the processors and the resources assigned to them. No matter how many processors we place in a system, or how many resources we assign to them, somehow, somewhere, a bottleneck is bound to develop. The best we can do, however, is make sure the slowest parts of the system are the ones that are used the least. This is the idea behind **Amdahl’s Law**. This law states that the performance enhancement possible with a given improvement is limited by the amount that the improved feature is used. The underlying premise is that every algorithm has a sequential part that ultimately limits the speedup that can be achieved by multiprocessor implementation.

If parallel machines and other non–von Neumann architectures give such huge increases in processing speed and power, why isn’t everyone using them everywhere? The answer lies in their programmability. Advances in operating systems that can utilize multiple cores have put these chips in laptops and desktops that we can buy today; however, true multiprocessor programming is more complex than both uniprocessor and multicore programming and requires people to think about problems in a different way, using new algorithms and programming tools.

One of these programming tools is a set of new programming languages. Most of our programming languages are von Neumann languages, created for the von Neumann architecture. Many common languages have been extended with special libraries to accommodate parallel programming, and many new languages have been designed specifically for the parallel programming environment. We have very few programming languages for the remaining (nonparallel) non–von Neumann platforms, and fewer people who really understand how to

program in these environments efficiently. Examples of non-von Neumann languages include Lucid (for dataflow) and QCL (Quantum Computation Language) for quantum computers, as well as VHDL or Verilog (languages used to program FPGAs). However, even with the inherent difficulties in programming parallel machines, we see in the next section that significant progress is being made.

1.11 PARALLELISM: ENABLER OF MACHINE INTELLIGENCE—DEEP BLUE AND WATSON

It is evident by our sidebar on the Mechanical Turk that chess playing has long been considered the ultimate demonstration of a “thinking machine.” The chessboard is a battlefield where human can meet machine on more-or-less equal terms—with the human always having the edge, of course. Real chess-playing computers have been around since the late 1950s. Over the decades, they gradually improved their hardware and software to eventually become formidable opponents for reasonably skilled players. The problem of *championship* chess playing, however, had long been considered so hard that many believed a machine could never beat a human Grandmaster. On May 11, 1997, a machine called Deep Blue did just that.

Deep Blue’s principal designers were IBM researchers Feng-hsiung Hsu, Thomas Anantharaman, and Murray Campbell. Reportedly costing more than \$6 million and taking six years to build, Deep Blue was a massively parallel system consisting of 30 RS/6000-based nodes supplemented with 480 chips built especially to play chess. Deep Blue included a database of 700,000 complete games with separate systems for opening and endgames. It evaluated 200 million positions per second on average. This enabled Deep Blue to produce a 12-move look ahead.

Having soundly beat an earlier version of Deep Blue, world chess champion Garry Kasparov was overwhelmingly favored to win a rematch starting May 3, 1997. At the end of five games, Kasparov and Deep Blue were tied, 2½ to 2½. Then Deep Blue quickly seized upon an error that Kasparov made early in the sixth game. Kasparov had no choice but to concede, thus making Deep Blue the first machine to ever defeat a chess Grandmaster.

With Deep Blue’s stunning win over Kasparov now in the history books, IBM Research manager Charles Lickel began looking for a new challenge. In 2004, Lickel was among the millions mesmerized by Ken Jennings’s unprecedented 74-game winning streak on the American quiz show, *Jeopardy!* As he watched Jennings win one match after another, Lickel dared to think that it was possible to build a machine that could win at *Jeopardy!* Moreover, he believed that IBM Research had the talent to build such a machine. He tapped Dr. David Ferrucci to lead the effort.

IBM scientists were in no rush to sign on to Lickel’s audacious project. They doubted—with good reason—that such a machine could be built. After all, creating Deep Blue was hard enough. Playing *Jeopardy!* is enormously more difficult than playing chess. In chess, the problem domain is clearly defined with fixed,

unambiguous rules, and a finite (although very large) solution space. *Jeopardy!* questions, on the other hand, cover a nearly infinite problem space compounded by the vagaries of human language, odd relations between concepts, puns, and vast amounts of unstructured factual information. For example, a *Jeopardy!* category could be titled “Doozy Twos” and relate to an African leader, an article of clothing, an Al Jolson song, and an ammunition size (Benjamin Tutu, tutu skirt, “Toot Toot Tootsie,” and .22 caliber). Whereas a human being has little trouble seeing the relationship (especially once the answer is revealed), computers are utterly baffled.

To make the game fair, Watson had to emulate a human player as closely as possible. No connection to the Internet or any other computers was permitted, and Watson was required to physically press a plunger to “buzz in” with an answer. However, Watson wasn’t programmed to process sound or images, so visual and strictly audio clues—such as musical selections—were not used during the match.

Once a clue was read, Watson initiated several parallel processes. Each process examined different aspects of the clue, narrowed the solution space, and formulated a hypothesis as to the answer. The hypothesis included a probability of its being correct. Watson selected the most likely of the hypotheses, or selected no hypothesis at all if the probability of correctness didn’t reach a predetermined threshold. Watson’s designers determined that if Watson were to attempt just 70% of the questions and respond correctly just 85% of the time, it would win the contest. No human players had ever done as well.

Using Watson’s algorithms, a typical desktop computer would need about two hours to come up with a good hypothesis. Watson had to do it in less than *three seconds*. It achieved this feat through a massively parallel architecture dubbed **DeepQA** (Deep Question and Answer). The system relied on 90 IBM POWER 750 servers. Each server was equipped with four POWER7 processors, and each POWER7 processor had eight cores, giving a total of 2880 processor cores. While playing *Jeopardy!*, each core had access to 16TB of main memory and 4TB of clustered storage.

Unlike Deep Blue, Watson could not be programmed to solve problems through brute force: The problem space was much too large. Watson’s designers, therefore, approached the situation just as a human being would: Watson “learned” by consuming terabytes of unstructured data from thousands of news sources, journals, and books. The DeepQA algorithms provided Watson with the ability to synthesize information—in a humanlike manner—from this universe of raw data. Watson drew inferences and made assumptions using hard facts and incomplete information. Watson could see information in context: The same question, in a different context, might well produce a different answer.

On the third day of its match, February 16, 2011, Watson stunned the world by soundly beating both reigning *Jeopardy!* champs, Ken Jennings and Brad Rutter. Watson’s winnings were donated to charity, but Watson’s service to humanity was only beginning. Watson’s ability to absorb and draw inferences

from pools of unstructured data made it a perfect candidate for medical school. Beginning in 2011, IBM, WellPoint, and Memorial Sloan-Kettering Cancer Center set Watson to work absorbing more than 600,000 pieces of medical evidence, and two million pages of text from 42 medical journals and oncology research documents. Watson's literature assimilation was supplemented with 14,700 hours of live training provided by WellPoint nurses. Watson was then given 25,000 test case scenarios and 1500 real-life cases from which it demonstrated that it had gained the ability to derive meaning from the mountain of complex medical data, some of which was in informal natural language—such as doctors' notes, patient records, medical annotations, and clinical feedback. Watson's *Jeopardy!* success has now been matched by its medical school success. Commercial products based on Watson technology, including “Interactive Care Insights for Oncology” and “Interactive Care Reviewer,” are now available. They hold the promise to improve the speed and accuracy of medical care for cancer patients.

Although Watson's applications and abilities have been growing, Watson's footprint has been shrinking. In the span of only a few years, system performance has improved by 240% with a 75% reduction in physical resources. Watson can now be run on a single POWER 750 server, leading some to claim that “Watson on a chip” is just around the corner.

In Watson, we have not merely seen an amazing *Jeopardy!* player or crack oncologist. What we have seen is the future of computing. Rather than people being trained to use computers, computers will train themselves to interact with people—with all their fuzzy and incomplete information. Tomorrow's systems will meet humans on human terms. As Dr. Ferrucci puts it, there simply is no other future for computers except to become like Watson. It just *has to* be this way.

CHAPTER SUMMARY

In this chapter, we have presented a brief overview of computer organization and computer architecture and shown how they differ. We also have introduced some terminology in the context of a fictitious computer advertisement. Much of this terminology will be expanded on in later chapters.

Historically, computers were simply calculating machines. As computers became more sophisticated, they became general-purpose machines, which necessitated viewing each system as a hierarchy of levels instead of one gigantic machine. Each layer in this hierarchy serves a specific purpose, and all levels help minimize the semantic gap between a high-level programming language or application and the gates and wires that make up the physical hardware. Perhaps the single most important development in computing that affects us as programmers is the introduction of the stored-program concept of the von Neumann machine. Although there are other architectural models, the von Neumann architecture is predominant in today's general-purpose computers.

FURTHER READING

We encourage you to build on our brief presentation of the history of computers. We think you will find this subject intriguing because it is as much about people as it is about machines. You can read about the “forgotten father of the computer,” John Atanasoff, in Mollenhoff (1988). This book documents the odd relationship between Atanasoff and John Mauchly, and recounts the open court battle of two computer giants, Honeywell and Sperry Rand. This trial ultimately gave Atanasoff his proper recognition.

For a lighter look at computer history, try the book by Rochester and Gantz (1983). Augarten’s (1985) illustrated history of computers is a delight to read and contains hundreds of hard-to-find pictures of early computers and computing devices. For a complete discussion of the historical development of computers, you can check out the three-volume dictionary by Cortada (1987). A particularly thoughtful account of the history of computing is presented in Ceruzzi (1998). If you are interested in an excellent set of case studies about historical computers, see Blaauw and Brooks (1997).

You will also be richly rewarded by reading McCartney’s (1999) book about the ENIAC, Chopsky and Leonsis’s (1988) chronicle of the development of the IBM PC, and Toole’s (1998) biography of Ada, Countess of Lovelace. Polachek’s (1997) article conveys a vivid picture of the complexity of calculating ballistic firing tables. After reading this article, you will understand why the army would gladly pay for anything that promised to make the process faster or more accurate. The Maxfield and Brown book (1997) contains a fascinating look at the origins and history of computing as well as in-depth explanations of how a computer works.

For more information on Moore’s Law, we refer the reader to Schaller (1997). For detailed descriptions of early computers as well as profiles and reminiscences of industry pioneers, you may wish to consult the *IEEE Annals of the History of Computing*, which is published quarterly. The Computer Museum History Center can be found online at www.computerhistory.org. It contains various exhibits, research, timelines, and collections. Many cities now have computer museums and allow visitors to use some of the older computers.

A wealth of information can be found at the websites of the standards-making bodies discussed in this chapter (as well as sites not discussed in this chapter). The IEEE can be found at www.ieee.org; ANSI at www.ansi.org; the ISO at www.iso.ch; the BSI at www.bsi-global.com; and the ITU-T at www.itu.int. The ISO site offers a vast amount of information and standards reference materials.

The WWW Computer Architecture Home Page at www.cs.wisc.edu/~arch/www/ contains a comprehensive index to computer architecture–related information. Many USENET newsgroups are devoted to these topics as well, including comp.arch and comp.arch.storage.

The entire May–June 2000 issue of MIT’s *Technology Review* magazine is devoted to architectures that may be the basis of tomorrow’s computers. Reading this issue will be time well spent. In fact, we could say the same of every issue.

For a truly unique account of human computers, we invite you to read Grier's *When Computers Were Human*. Among other things, he presents a stirring account of the human computers who drove the mathematical tables project under the Depression-era Works Progress Administration (WPA). The contributions made by these "table factories" were crucial to America's victory in World War II. A shorter account of this effort can also be found in Grier's 1998 article that appears in the *IEEE Annals of the History of Computing*.

The entire May–June 2012 issue of the *IBM Journal of Research and Development* is dedicated to the building of Watson. The two articles by Ferrucci and Lewis give great insight into the challenges and triumphs of this groundbreaking machine. The IBM whitepaper, "Watson—A System Designed for Answers," provides a nice summary of Watson's hardware architecture. Feng-hsiung Hsu gives his first-person account of the building of Deep Blue in *Behind Deep Blue: Building the Computer that Defeated the World Chess Champion*. Readers interested in the Mechanical Turk can find more information in the book of the same name by Tom Standage.

REFERENCES

- Augarten, S. *Bit by Bit: An Illustrated History of Computers*. London: Unwin Paperbacks, 1985.
- Blaauw, G., & Brooks, F. *Computer Architecture: Concepts and Evolution*. Reading, MA: Addison-Wesley, 1997.
- Ceruzzi, P. E. *A History of Modern Computing*. Cambridge, MA: MIT Press, 1998.
- Chopsky, J., & Leonsis, T. *Blue Magic: The People, Power and Politics Behind the IBM Personal Computer*. New York: Facts on File Publications, 1988.
- Cortada, J. W. *Historical Dictionary of Data Processing*, Volume 1: *Biographies*; Volume 2: *Organization*; Volume 3: *Technology*. Westport, CT: Greenwood Press, 1987.
- Ferrucci, D. A., "Introduction to 'This is Watson.'" *IBM Journal of Research and Development* 56:3/4, May–June 2012, pp. 1:1–1:15.
- Grier, D. A. "The Math Tables Project of the Work Projects Administration: The Reluctant Start of the Computing Era." *IEEE Annals of the History of Computing* 20:3, July–Sept. 1998, pp. 33–50.
- Grier, D. A. *When Computers Were Human*. Princeton, NJ: Princeton University Press, 2007.
- Hsu, F.-h. *Behind Deep Blue: Building the Computer that Defeated the World Chess Champion*. Princeton, NJ: Princeton University Press, 2006.
- IBM. "Watson—A System Designed for Answers: The future of workload optimized systems design." February 2011. <ftp://public.dhe.ibm.com/common/ssi/ecm/en/pow03061usen/POW-03061USEN.PDF>. Retrieved June 4, 2013.
- Lewis, B. L. "In the game: The interface between Watson and *Jeopardy!*" *IBM Journal of Research and Development* 56:3/4, May–June 2012, pp. 17:1–17:6.
- Maguire, Y., Boyden III, E. S., & Gershenfeld, N. "Toward a Table-Top Quantum Computer." *IBM Systems Journal* 39:3/4, June 2000, pp. 823–839.
- Maxfield, C., & Brown, A. *Bebop BYTES Back (An Unconventional Guide to Computers)*. Madison, AL: Doone Publications, 1997.

- McCartney, S. *ENIAC: The Triumphs and Tragedies of the World's First Computer*. New York: Walker and Company, 1999.
- Mollenhoff, C. R. *Atanasoff: The Forgotten Father of the Computer*. Ames, IA: Iowa State University Press, 1988.
- Polachek, H. "Before the ENIAC." *IEEE Annals of the History of Computing* 19:2, June 1997, pp. 25–30.
- Rochester, J. B., & Gantz, J. *The Naked Computer: A Layperson's Almanac of Computer Lore, Wizardry, Personalities, Memorabilia, World Records, Mindblowers, and Tomfoolery*. New York: William A. Morrow, 1983.
- Schaller, R. "Moore's Law: Past, Present, and Future." *IEEE Spectrum*, June 1997, pp. 52–59.
- Stallings, W. *Operating Systems: Internals and Design Principles*, 7th ed. Upper Saddle River, NJ: Prentice Hall, 2012.
- Standage, T. *The Turk: The Life and Times of the Famous Eighteenth-Century Chess-Playing Machine*. New York: Berkley Trade, 2003.
- Tanenbaum, A. *Structured Computer Organization*, 6th ed. Upper Saddle River, NJ: Prentice Hall, 2013.
- Toole, B. A. *Ada, the Enchantress of Numbers: Prophet of the Computer Age*. Mill Valley, CA: Strawberry Press, 1998.
- Waldrop, M. M. "Quantum Computing." *MIT Technology Review* 103:3, May/June 2000, pp. 60–66.

REVIEW OF ESSENTIAL TERMS AND CONCEPTS

1. What is the difference between computer organization and computer architecture?
2. What is an ISA?
3. What is the importance of the Principle of Equivalence of Hardware and Software?
4. Name the three basic components of every computer.
5. To what power of 10 does the prefix giga- refer? What is the (approximate) equivalent power of 2?
6. To what power of 10 does the prefix micro- refer? What is the (approximate) equivalent power of 2?
7. What unit is typically used to measure the speed of a computer clock?
8. What are the distinguishing features of tablet computers?
9. Name two types of computer memory.
10. What is the mission of the IEEE?
11. What is the full name of the organization that uses the initials ISO? Is ISO an acronym?
12. ANSI is the acronym used by which organization?
13. What is the name of the Swiss organization that devotes itself to matters concerning telephony, telecommunications, and data communications?
14. Who is known as the father of computing, and why?
15. What was the significance of the punched card?

16. Name two driving factors in the development of computers.
17. What is it about the transistor that made it such a great improvement over the vacuum tube?
18. How does an integrated circuit differ from a transistor?
19. Explain the differences between SSI, MSI, LSI, and VLSI.
20. What technology spawned the development of microcomputers? Why?
21. What is meant by an “open architecture”?
22. State Moore’s Law.
23. How is Rock’s Law related to Moore’s Law?
24. Name and explain the seven commonly accepted layers of the Computer Level Hierarchy. How does this arrangement help us to understand computer systems?
25. How does the term *abstraction* apply to computer organization and architecture?
26. What was it about the von Neumann architecture that distinguished it from its predecessors?
27. Name the characteristics present in von Neumann architecture.
28. How does the fetch-decode-execute cycle work?
29. What is a multicore processor?
30. What are the key characteristics of Cloud computing?
31. What are the three types of Cloud computing platforms?
32. What are the main challenges of Cloud computing from a provider perspective as well as a consumer perspective?
33. What are the advantages and disadvantages of service-oriented computing?
34. What is meant by parallel computing?
35. What is the underlying premise of Amdahl’s Law?
36. What makes Watson so different from traditional computers?

EXERCISES

- ♦ 1. In what ways are hardware and software different? In what ways are they the same?
2. a) How many milliseconds (ms) are in 1 second?
- b) How many microseconds (μ s) are in 1 second?
- c) How many nanoseconds (ns) are in 1 millisecond?
- d) How many microseconds are in 1 millisecond?
- e) How many nanoseconds are in 1 microsecond?
- f) How many kilobytes (KB) are in 1 gigabyte (GB)?
- g) How many kilobytes are in 1 megabyte (MB)?

- h) How many megabytes are in 1 gigabyte?
 - i) How many bytes are in 20 megabytes?
 - j) How many kilobytes are in 2 gigabytes?
- ◆ 3. By what order of magnitude is something that runs in nanoseconds faster than something that runs in milliseconds?
 4. Pretend you are ready to buy a new computer for personal use. First, take a look at ads from various magazines and newspapers and list terms you don't quite understand. Look up these terms and give a brief written explanation. Decide what factors are important in your decision as to which computer to buy and list them. After you select the system you would like to buy, identify which terms refer to hardware and which refer to software.
 5. Makers of tablet computers continually work within narrow constraints on cost, power consumption, weight, and battery life. Describe what you feel would be the perfect tablet computer. How large would the screen be? Would you rather have a longer-lasting battery, even if it means having a heavier unit? How heavy would be too heavy? Would you rather have low cost or fast performance? Should the battery be consumer replaceable?
 6. Pick your favorite computer language and write a small program. After compiling the program, see if you can determine the ratio of source code instructions to the machine language instructions generated by the compiler. If you add one line of source code, how does that affect the machine language program? Try adding different source code instructions, such as an add and then a multiply. How does the size of the machine code file change with the different instructions? Comment on the result.
 7. Respond to the idea presented in Section 1.5: If invented today, what name do you think would be given to the computer? Give at least one good reason for your answer.
 8. Briefly explain two breakthroughs in the history of computing.
 9. Would it be possible to fool people with an automaton like the Mechanical Turk today? If you were to try to create a Turk today, how would it differ from the eighteenth-century version?
 - ◆ 10. Suppose a transistor on an integrated circuit chip were 2 microns in size. According to Moore's Law, how large would that transistor be in 2 years? How is Moore's Law relevant to programmers?
 11. What circumstances helped the IBM PC become so successful?
 12. List five applications of personal computers. Is there a limit to the applications of computers? Do you envision any radically different and exciting applications in the near future? If so, what?
 13. In the von Neumann model, explain the purpose of the:
 - a) processing unit
 - b) program counter

14. Under the von Neumann architecture, a program and its data are both stored in memory. It is therefore possible for a program, thinking that a memory location holds a piece of data when it actually holds a program instruction, to accidentally (or on purpose) modify itself. What implications does this present to you as a programmer?
15. Explain why modern computers consist of multiple levels of virtual machines.
16. Explain the three main types of Cloud computing platforms.
17. What are the challenges facing organizations that wish to move to a Cloud platform? What are the risks and benefits?
18. Does Cloud computing eliminate all of an organization's concerns about its computing infrastructure?
19. Explain what it means to "fetch" an instruction.
20. Read a popular local newspaper and search through the job openings. (You can also check some of the more popular online career sites.) Which jobs require specific hardware knowledge? Which jobs imply knowledge of computer hardware? Is there any correlation between the required hardware knowledge and the company or its location?
21. List and describe some common uses and some not-so-common uses of computers in business and other sectors of society.
22. The technologist's notion of Moore's Law is that the number of transistors per chip doubles approximately every 18 months. In the 1990s, Moore's Law started to be described as the doubling of microprocessor power every 18 months. Given this new variation of Moore's Law, answer the following:
 - a) After successfully completing your computer organization and architecture class, you have a brilliant idea for a new chip design that would make a processor six times faster than the fastest ones on the market today. Unfortunately, it will take you four and a half years to save the money, create the prototype, and build a finished product. If Moore's Law holds, should you spend your money developing and producing your chip or invest in some other venture?
 - b) Suppose we have a problem that currently takes 100,000 hours of computer time using current technology to solve. Which of the following would give us the solution first: (1) Replace the algorithm used in the current solution with one that runs twice as fast and run it on the current technology, or (2) Wait 3 years, assuming Moore's Law doubles the performance of a computer every 18 months, and find the solution using the current algorithm with the new technology?
23. What are the limitations of Moore's Law? Why can't this law hold forever? Explain.
24. What are some technical implications of Moore's Law? What effect does it have on your future?
25. Do you share Dr. Ferrucci's opinion that all computers will become like Watson someday? If you had a tablet-sized Watson, what would you do with it?

