"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 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 tradeoffs, 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 area 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 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:** *Anything that can be done with software can also be done with hardware, and anything that can be done with hardware can also be done with 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.

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

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

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 book 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 "64MB SDRAM," "64-bit PCI sound card" and "32KB 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 system is able to serve your needs. As we progress through this book, 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. So that computer people can tell each other how big something is, or how fast something is, they must use the same units of measure. When we want to talk about how big some computer thing is, we speak of it in terms of thousands, millions, billions, or trillions of characters. The prefixes for terms are given in the left side of Figure 1.2. In computing systems, as you shall see, powers of 2 are often more important than powers of 10, but it is easier for people to understand powers of 10. Therefore, these prefixes are given in both powers of 10 and powers of 2. Because 1,000 is close in value to 2^{10} (1,024), we can approximate powers of 10 by powers of 2. Prefixes used in system metrics are often applied where the underlying base system is base 2, not base 10. For example, a

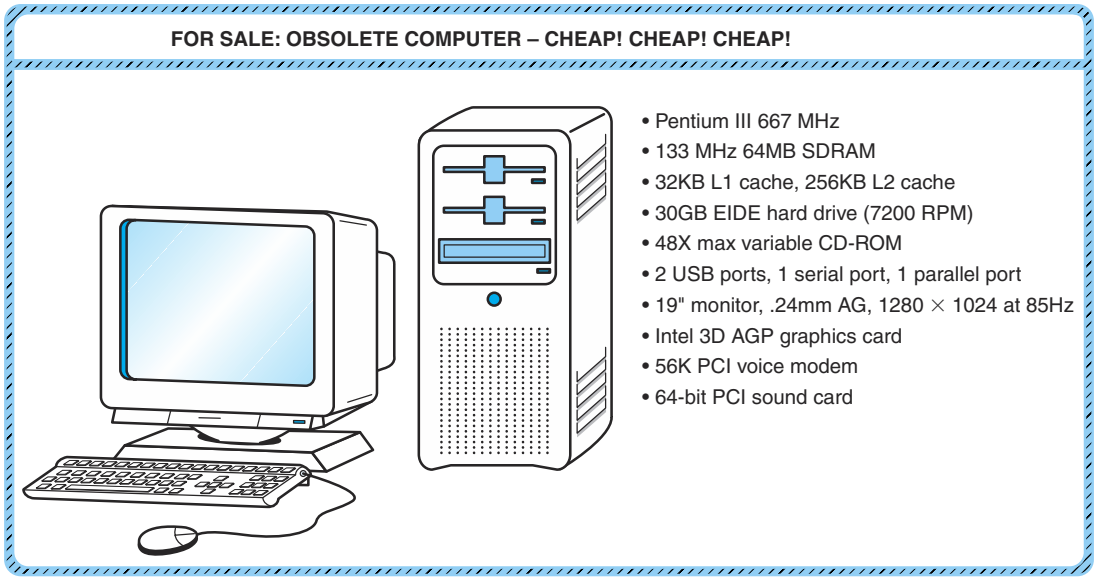


FIGURE 1.1 A Typical Computer Advertisement

Kilo- (K)	(1 thousand = $10^3 \approx 2^{10}$)	Milli- (m)	(1 thousandth = $10^{-3} \approx 2^{-10}$)
Mega- (M)	(1 million = $10^6 \approx 2^{20}$)	Micro- (μ)	(1 millionth = $10^{-6} \approx 2^{-20}$)
Giga- (G)	(1 billion = $10^9 \approx 2^{30}$)	Nano- (n)	(1 billionth = $10^{-9} \approx 2^{-30}$)
Tera- (T)	(1 trillion = $10^{12} \approx 2^{40}$)	Pico- (p)	(1 trillionth = $10^{-12} \approx 2^{-40}$)
Peta- (P)	(1 quadrillion = $10^{15} \approx 2^{50}$)	Femto- (f)	(1 quadrillionth = $10^{-15} \approx 2^{-50}$)

FIGURE 1.2 Common Prefixes Associated with Computer Organization and Architecture

kilobyte (1KB) of memory is *typically* 1,024 bytes of memory rather than 1,000 bytes of memory. However, a 1GB disk drive might actually be 1 billion bytes instead of 2^{30} (approximately 1.7 billion). You should always read the manufacturer's fine print just to make sure you know exactly what 1K, 1KB, or 1G represents.

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 Figure 1.2. Notice that the fractional prefixes have exponents that are the reciprocal of the prefixes on the left side of the figure. 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 is the part of a computer that actually executes program instructions; it is the brain of the system. The microprocessor in the ad is a Pentium III, operating at 667MHz. 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*. Because computer system clocks generate millions of pulses per second, we say that they operate in the *megahertz (MHz)* range. Many computers today operate in the *gigahertz* 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 667 million cycles per second, so the seller says that it runs at 667MHz.

The fact that this microprocessor runs at 667MHz, however, doesn't necessarily mean that it can execute 667 million instructions every second, or, equivalently, that every instruction requires 1.5 nanoseconds to execute. Later in this book, 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.

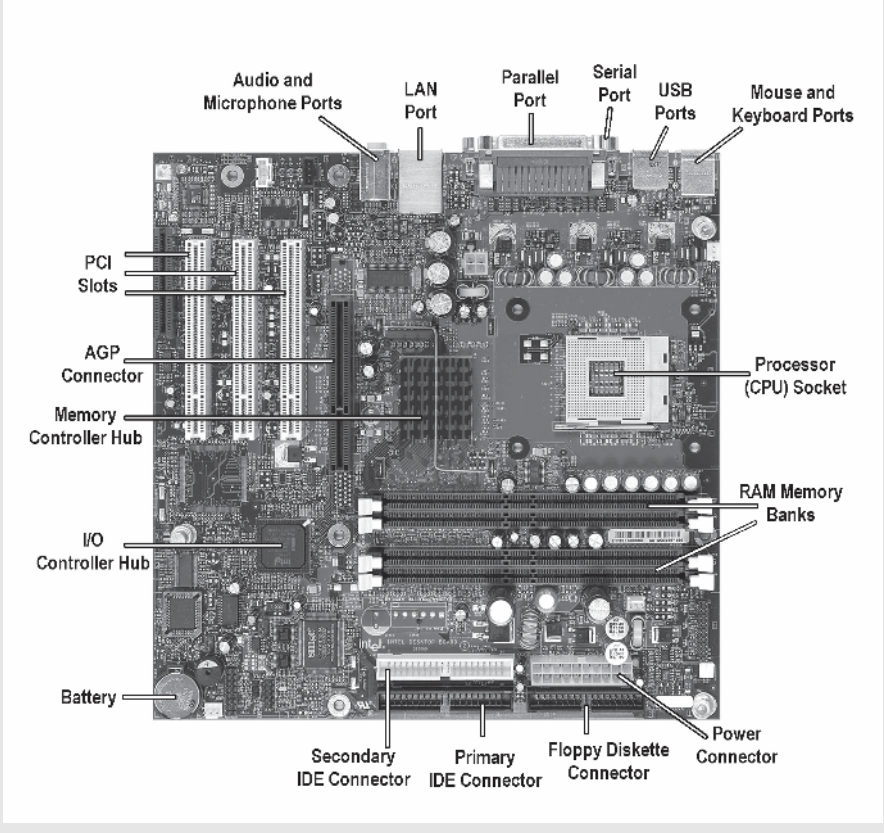
The next thing that we see in the ad is "133MHz 64MB SDRAM." The 133MHz refers to the speed of the system *bus*, which is a group of wires that moves data and instructions to various places within the computer. Like the microprocessor, the speed of the bus is also measured in MHz. Many computers have 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 64 megabytes (MB), or about 64 million characters. Memory capacity not only determines the size of the programs that 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 their 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. At this writing, SDRAM bus synchronization is possible only with buses running at 200MHz and below. Newer memory technologies such as RDRAM (Rambus DRAM) and SLDRAM (SyncLink DRAM) are required for systems running faster buses.

A Look Inside a Computer

Have you even 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.



Courtesy of Intel Corporation

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 perhaps a floppy drive and CD-ROM or DVD 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 of the components in the

computer, including the CPU, and RAM and ROM memory, 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 Intel D850 motherboard with the more important components labeled.

The I/O ports at the top of the board allow the computer to communicate with the outside world. The I/O controller hub allows all connected devices to function without conflict. The PCI slots allow for expansion boards belonging to various PCI devices. The AGP connector is for plugging in the AGP graphics card. There are two RAM memory banks and a memory controller hub. There is no processor plugged into this motherboard, but we see the socket where the CPU is to be placed. All computers have an internal battery, as seen at the lower left-hand corner. This motherboard has two IDE connector slots, and one floppy disk controller. The power supply plugs into the power connector.

A note of caution regarding looking inside the box: There are many safety considerations involved with removing the cover for both you and your computer. 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, "32KB L1 cache, 256KB 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 30GB, which is fairly impressive. 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 RPM (revolutions per minute). 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 disk interface called *EIDE*, or *enhanced integrated drive electronics*. EIDE is a cost-effective hardware interface for mass storage devices. EIDE contains special circuits that allow it to enhance a computer's connectivity, speed, and memory capability. Most 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 three different ports with the line, "2 USB ports, 1 serial port, 1 parallel port." Most desktop computers come with two kinds of data ports: serial ports and parallel ports. Serial ports transfer data by sending a series of electrical pulses across one or two data lines. Parallel ports use at least eight data lines, which are energized simultaneously to transmit data. Our advertised system also comes equipped with a special serial connection called a *USB* (*universal serial bus*) port. USB is a popular external bus that supports *Plug-and-Play* (the ability to configure devices automatically) as well as *hot plugging* (the ability to add and remove devices while the computer is running).

Some systems augment their main bus with dedicated I/O buses. *Peripheral Component Interconnect* (*PCI*) is one such I/O bus that supports the connection of multiple peripheral devices. PCI, developed by the Intel Corporation, operates at high speeds and also supports Plug-and-Play. There are two PCI devices mentioned in the ad. The PCI modem allows the computer to connect to the Internet. (We discuss modems in detail in Chapter 11.) The other PCI device is a sound card, which contains components needed by the system's stereo speakers. You will learn more about different kinds of I/O, I/O buses, and disk storage in Chapter 7.

After telling us about the ports in the advertised system, the ad supplies us with some specifications for the monitor by saying, "19" monitor, .24mm AG, 1280 $\times$ 1024 at 85Hz." 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. The monitor in the ad supports a *refresh rate* of 85Hz. This means that the image displayed on the monitor is repainted 85 times a second. If the refresh rate is too slow, the screen may exhibit an annoying jiggle or wavelike behavior. The eyestrain caused by a wavy display makes people tire easily; some people may even experience headaches after periods of prolonged use. Another source of eyestrain is poor resolution. A higher-resolution monitor makes for better viewing and finer graphics. Resolution is determined by the *dot pitch* of the monitor, which is the distance between a dot (or pixel) and the closest dot of the same color. The smaller the dot, the sharper the image. In this case, we have a 0.28 millimeter

(mm) dot pitch supported by an *AG* (*aperture grill*) display. Aperture grills direct the electron beam that paints the monitor picture on the phosphor coating inside the glass of the monitor. AG monitors produce crisper images than the older shadow mask technology. This monitor is further supported by an *AGP* (*accelerated graphics port*) graphics card. This is a graphics interface designed by Intel specifically for 3D graphics.

In light of the preceding discussion, you may be wondering why monitor dot pitch can't be made arbitrarily small to give picture perfect resolution. The reason is that the refresh rate is dependent on the dot pitch. Refreshing 100 dots, for example, requires more time than refreshing 50 dots. A smaller dot pitch requires more dots to cover the screen. The more dots to refresh, the longer it takes for each refresh cycle. Experts recommend a refresh rate of at least 75Hz. The 85Hz refresh rate of the advertised monitor is better than the minimum recommendation by 10Hz (about 13%).

Although we cannot delve into all of the brand-specific components available, after completing this book, 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 impact on your decision to access array elements in row versus column-major order.

Throughout this book, 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 that you gain from this book will ultimately serve as a springboard for your continuing studies within the vast and exciting fields of computer organization and architecture.

1.4 STANDARDS ORGANIZATIONS

Suppose you decide that you'd like to have one of those nifty new .28mm dot pitch AG monitors. You figure that 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 that you can buy your monitor anywhere and it will probably work fine on your system. You can make this assumption because computer equipment manu-

facturers 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 Electronic 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 within 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 *CEN* (*Comite Europeen 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 over 2,800 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 book, we mention official standards designations where appropriate. Definitive information concerning many of these standards can be

found in excruciating detail on the Web site 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 50-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 of 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 books have been written on this subject and even they leave their readers wanting for more detail. If we have piqued your interest, we refer you to look at 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 accu-

rate. 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 over 50 years.

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.

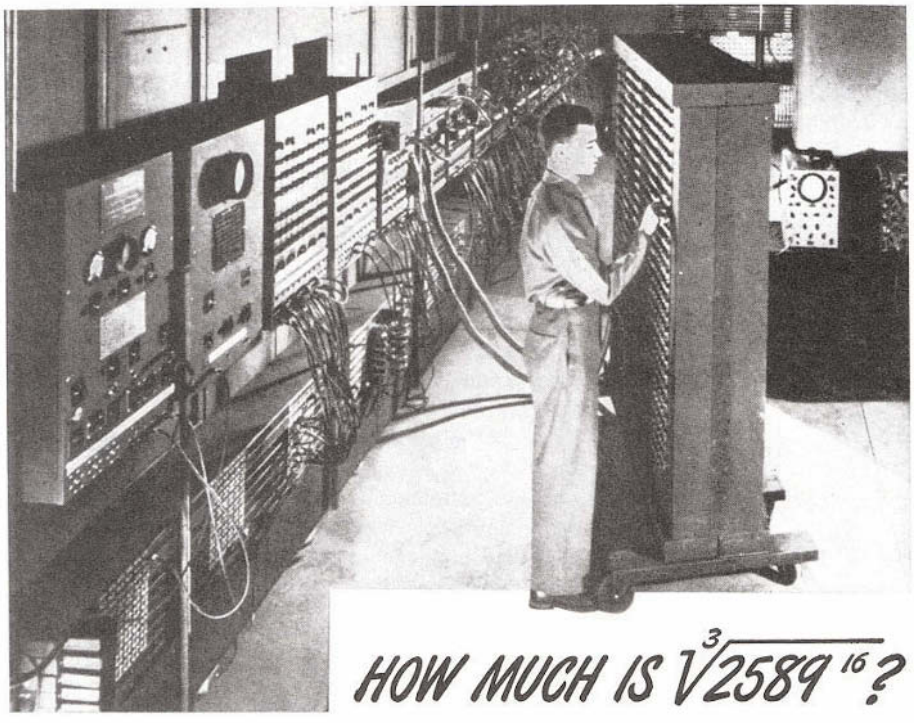
John Atanasoff (1904–1995) has been credited with the construction of the first completely electronic computer. The Atanasoff Berry Computer (ABC) was a binary machine built from vacuum tubes. Because this system was built specifi-

cally 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 principle 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 1,800 square feet of floor space, weighed 30 tons, and consumed 174 kilowatts of power. The ENIAC had a memory capacity of about 1,000 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 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 United States 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.



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 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.

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

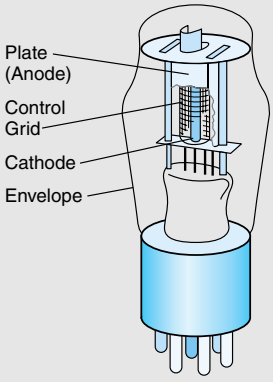
A GOOD JOB FOR YOU

U. S. Army

CHOOSE THIS FINE PROFESSION NOW!

U.S. Army, 1946

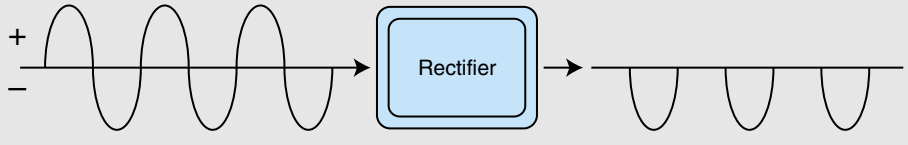
What Is a Vacuum Tube?



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 gasses, such as mercury vapor, which can provide desirable electrical behavior.

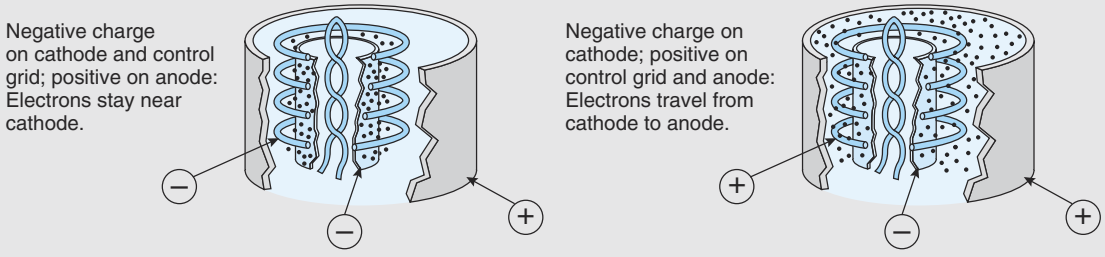
The electrical phenomenon that makes tubes work was discovered by Thomas A. Edison in 1883 while he was trying to find ways to keep the filaments of his light bulbs from burning away (or oxidizing) a few minutes after electrical current was applied. Edison reasoned correctly that one way to prevent filament 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*.

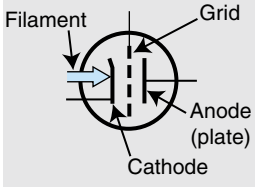


The diode was well suited for changing alternating current into direct current, but the greatest power of the electron tube was yet to be discovered. In

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.

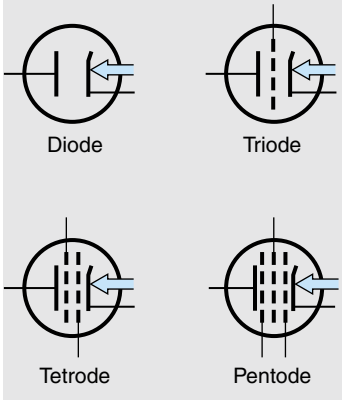


When DeForest patented his device, he called it an *audion tube*. It was later known as a *triode*. The schematic symbol for the triode is shown at the left.



A triode can act either as a switch or as 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 that 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 one-half inch (1.5cm) in diameter. Equivalent full-sized diodes, triodes, and pentodes were just 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 of electrical power were required to heat the cathodes of these devices. To prevent a meltdown, this heat had to be removed from the system as quickly as possible. Power consumption and heat dissipation could be reduced by running the cathode heaters at lower voltages, but this reduced the already slow switching speed of the tube. Despite their limitations and power consumption, vacuum tube computer

systems, both analog and digital, served their purpose for many years and are the architectural foundation for all modern computer systems.

Although decades have passed since the last vacuum tube computer was manufactured, vacuum tubes are still used in audio amplifiers. These "high-end" amplifiers are favored by musicians who believe that tubes provide a resonant and pleasing sound unattainable by solid-state devices.

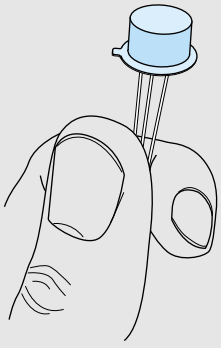
1.5.3 The Second Generation: Transistorized Computers (1954–1965)

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.

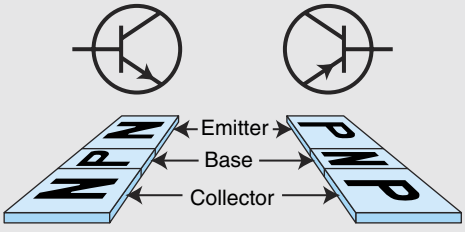
What Is a Transistor?

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. Because electrons are better behaved in a solid medium than in the open void of a vacuum tube, they need no 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.

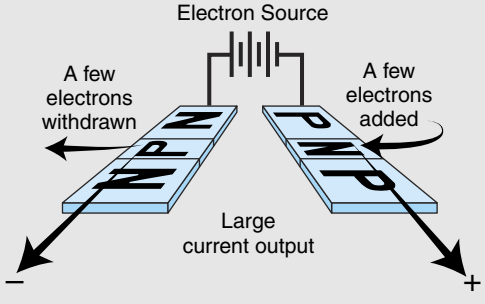
Boron, aluminum, and gallium can be found to the left of silicon and germanium on the Periodic Chart. Because they lie to the left of silicon and germanium, they have one less electron in their outer electron shell, or *valence*. So if you add a small amount of aluminum to silicon, the silicon ends up 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.



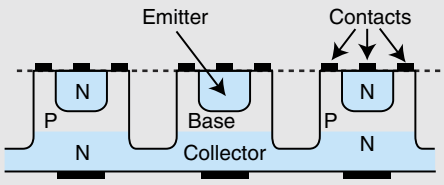
Similarly, if we add a little boron, arsenic, or gallium to silicon, we'll have extra electrons in valences of the silicon crystals. This gives us an *N-type* material. A small amount of current will flow through the N-type material if we provide the loosely bound electrons in the N-type material with 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 that 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 emitter.

The figure to 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 beginning 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



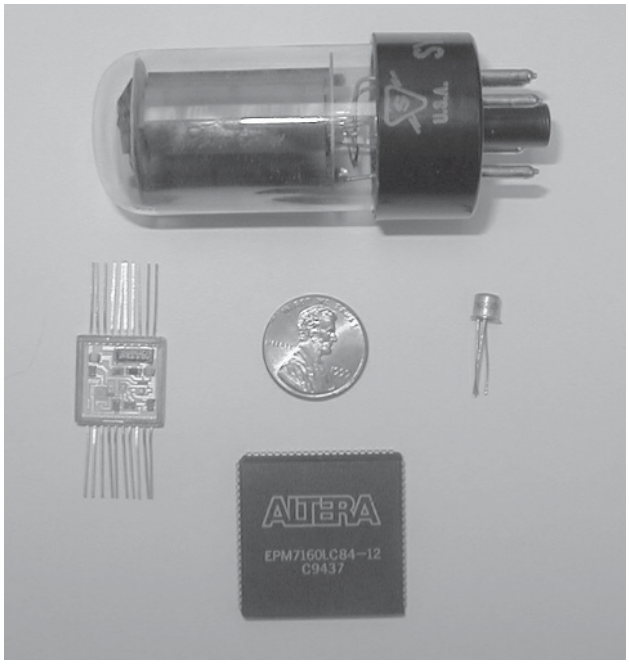
smaller and run cooler than their vacuum tube predecessors. The ultimate miniaturization, however, is not realized by replacing individual triodes with discrete transistors, but in shrinking entire circuits onto one piece of silicon.

Integrated circuits, or *chips*, contain hundreds to millions 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 of the layers have been etched. The resulting peaks and valleys of P- and N-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 where all of 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 of 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.



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

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 over 160 million instructions per second and could support 8 megabytes of memory.

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 1,000 components per chip; LSI (large scale integration), in which there are 1,000 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.

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 1,800 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 as well as 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 of the ground only after promising to deliver it within a year, a seemingly impossible feat.

Estridge knew that the only way that 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 16 megabytes 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 128 kilobytes 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 that 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 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 generation, and whatever those eras will bring.

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 law 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 cur-

rent 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 has seen the price tag of new chip facilities escalate from about $12,000 in 1968 to $12 million in the late 1990s. 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, 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

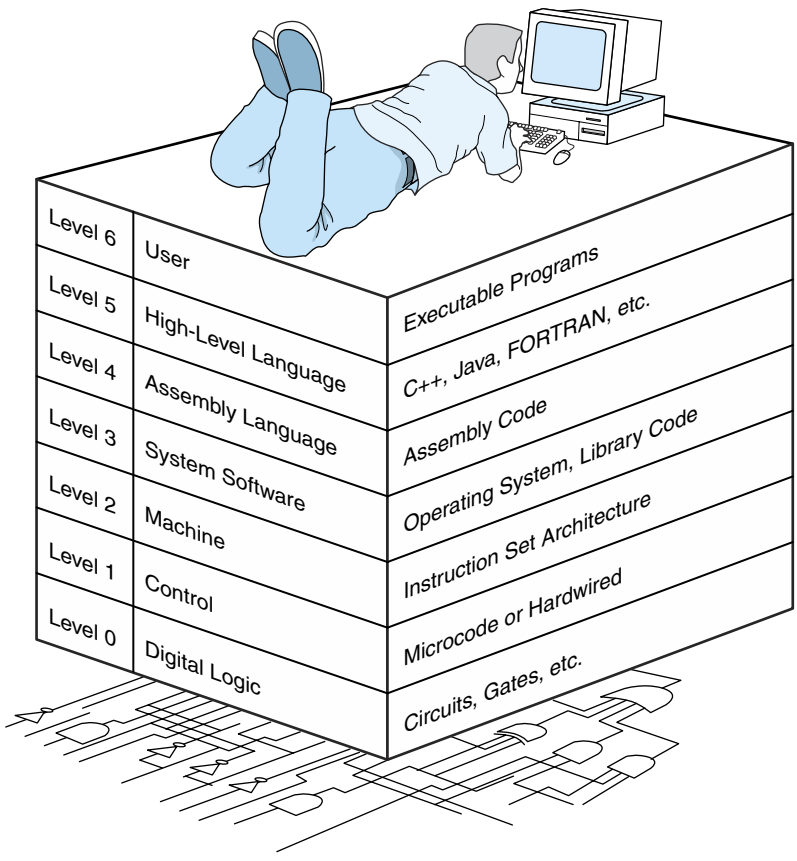


FIGURE 1.3 The Abstract Levels of Modern Computing Systems

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 (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, 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 lan-

guages 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 instruction set architectures 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 of 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 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 1.4 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 into registers within the CPU.
4. The ALU executes the instruction and places the results in registers or memory.

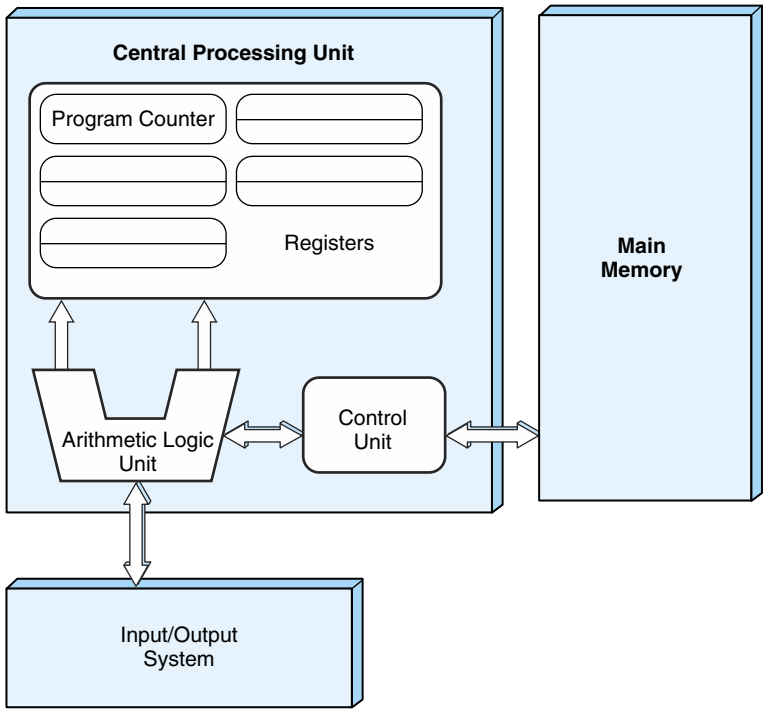


FIGURE 1.4 The von Neumann Architecture

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 1.5. The data bus moves data from main memory to the CPU registers (and vice versa). The address bus holds the 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.

1.8 NON–VON NEUMANN MODELS

Until recently, almost all general-purpose computers followed the von Neumann design. 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

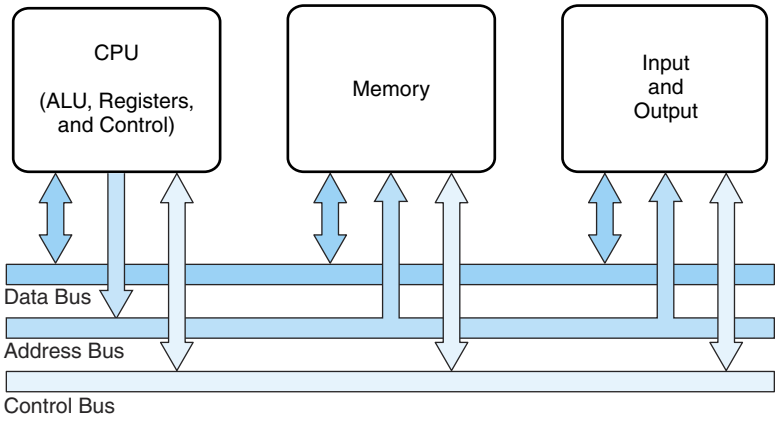


FIGURE 1.5 The Modified von Neumann Architecture, Adding a System Bus

by the need to maintain compatibility with von Neumann systems are free to use many different models of computing. 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), genetic algorithms (exploiting ideas from biology and DNA evolution), quantum computation (previously discussed), and parallel computers. Of these, parallel computing is currently the most popular.

Today, parallel processing solves some of our biggest problems in much the same way as 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 a 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 over 1,000 processors. Finally, in 1999, IBM announced the construction of a supercomputer called the Blue Gene. This massively parallel computer contains over 1 million processors, each with its own dedicated memory. Its first task is to analyze the behavior of protein molecules.

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 are placed in a system, or how many resources are assigned to them, somehow, somewhere, a bottleneck is bound to develop. The best that we can do to remedy this is to make sure that 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 fea-

ture 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.

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 that 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 & Brooks (1997).

You will also be richly rewarded by reading McCartney's (1999) book about the ENIAC, Chopsky and Leonsis' (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 Web sites of the standards-making bodies discussed in this chapter (as well as the 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.

REFERENCES

Augarten, Stan. *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, Paul E. *A History of Modern Computing.* Cambridge, MA: MIT Press, 1998.

Chopsky, James, & Leonsis, Ted. *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.

Maguire, Yael, Boyden III, Edward S., and Gershenfeld, Neil. "Toward a Table-Top Quantum Computer." *IBM Systems Journal 39*: 3/4 (June 2000), pp. 823–839.

Maxfield, Clive, & Brown, A. *Bebop BYTES Back* (*An Unconventional Guide to Computers*). Madison, AL: Doone Publications, 1997.

McCartney, Scott. *ENIAC: The Triumphs and Tragedies of the World's First Computer*. New York: Walker and Company, 1999.

Mollenhoff, Clark R. *Atanasoff: The Forgotten Father of the Computer*. Ames, IA: Iowa State University Press, 1988.

Polachek, Harry. "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.

Tanenbaum, A. *Structured Computer Organization*, 4th ed. Upper Saddle River, NJ: Prentice Hall, 1999.