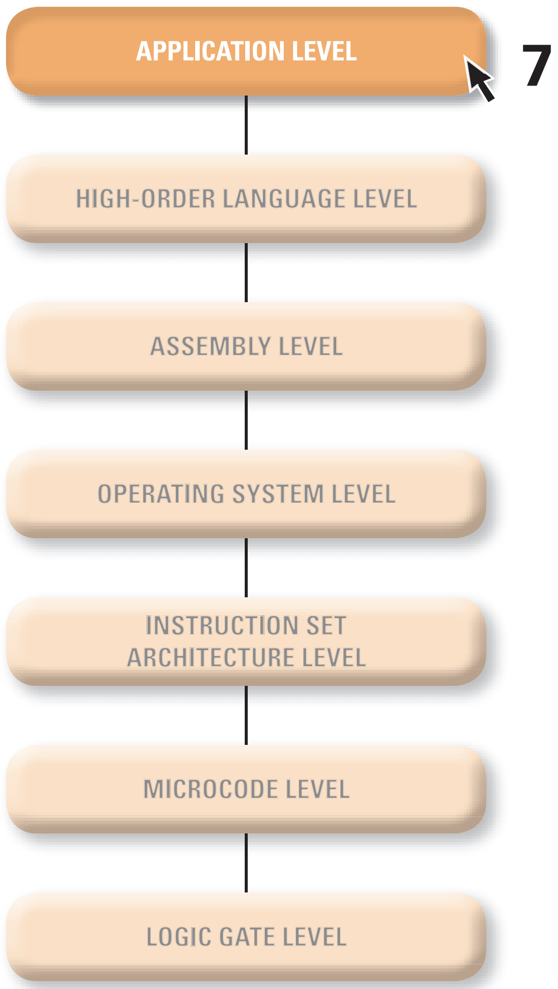
LEVEL
7

Application



CHAPTER 1

Computer Systems

TABLE OF CONTENTS

The fundamental question of computer science

The fundamental question of computer science is: What can be automated? Just as the machines developed during the Industrial Revolution automated manual labor, computers automate the processing of information. When electronic computers were developed in the 1940s, their designers built them to automate the solution of mathematical problems. Since then, however, computers have applications as diverse as financial accounting, cinema production, and smartphones. The spread of computers is so relentless that new areas of computer automation appear almost daily.

The purpose of this text is to show how the computer automates the processing of information. Everything the computer does, you could do in principle. The major difference between computer and human execution of a job is that the computer can perform its tasks blindingly fast. However, to harness its speed, people must instruct, or program, the computer.

Programming languages

The nature of computers is best understood by learning how to program the machine. Programming requires that you learn a programming language. Before plunging into the details of studying a programming language, this chapter introduces the concept of *abstraction*, the theme on which this text is based. It then describes the hardware and software components of a computer system and concludes with a description of a database system as a typical application.

1.1 Levels of Abstraction

The concept of levels of abstraction is pervasive in the arts as well as in the natural and applied sciences. A complete definition of abstraction is multifaceted and, for our purposes, includes the following parts:

Definition of abstraction

- Suppression of detail to show the essence of the matter
- An outline structure
- Division of responsibility through a chain of command
- Subdivision of a system into smaller subsystems

The theme of this text is the application of abstraction to computer science. It begins, however, by considering levels of abstraction in areas other than computer science. The analogies drawn from these areas expand on the four parts of the preceding definition of abstraction and apply to computer systems as well.

Three common graphic representations of levels of abstraction are (1) level diagrams, (2) nesting diagrams, and (3) hierarchy, or tree, diagrams, as shown in FIGURE 1.1. This section considers each of these representations of abstraction and shows how they relate to the analogies. The three diagrams also apply to levels of abstraction in computer systems throughout this text.

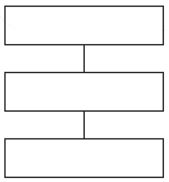
Figure 1.1(a) shows a *level diagram* as a set of boxes arranged vertically. The top box represents the highest level of abstraction, and the bottom box represents the lowest. The number of levels of abstraction depends on the system to be described. This figure would represent a system with three levels of abstraction.

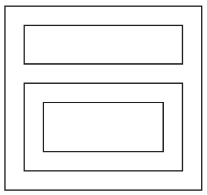
Figure 1.1(b) shows a *nesting diagram*. Like the level diagram, a nesting diagram is a set of boxes. It always consists of one large outer box with the rest of the boxes nested inside it. In the figure, two boxes are nested immediately inside the one large outer box. The lower of these two boxes has one box nested, in turn, inside it. The outermost box of a nesting diagram corresponds to the top box of a level diagram. The nested boxes correspond to the lower boxes of a level diagram.

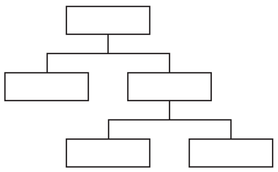
In a nesting diagram, none of the boxes overlap. That is, nesting diagrams never contain boxes whose boundaries intersect the boundaries of other boxes. A box is always completely enclosed within another box.

Figure 1.1(c) shows the third graphic representation of levels of abstraction, a *hierarchy*, or *tree*, *diagram*. In a tree, the big limbs branch off the trunk, the smaller limbs branch off the big limbs, and so on. The leaves are at the end of the chain, attached to the smallest branches. Tree diagrams such as Figure 1.1(c) have the trunk at the top instead of the bottom. Each box is called a *node*, with the single node at the top called the *root*. A node with no connections to a lower level is a *leaf*. This figure is a tree with one root node and three leaves. The top node in a hierarchy diagram corresponds to the top box of a level diagram.

FIGURE 1.1
The three graphic representations of levels of abstraction.


(a) A level diagram.


(b) A nesting diagram.


(c) A hierarchy, or tree, diagram.

Abstraction in Art

Henri Matisse was a major figure in the history of modern art. In 1909, he produced a bronze sculpture of a woman's back titled *The Back I*. Four years later, he created a work of the same subject but with a simpler rendering of the form, titled *The Back II*. After 4 more years, he created *The Back III*, followed by *The Back IV* 13 years later. FIGURE 1.2 shows the four sculptures.

A striking feature of the works is the elimination of detail as the artist progressed from one piece to the next. The contours of the back become less distinct in the second sculpture. The fingers of the right hand are hidden in the third. The hips are barely discernible in the fourth, which is the most abstract.

Matisse strove for expression. He deliberately suppressed visual detail in order to express the essence of the subject. In 1908, he wrote:

> In a picture, every part will be visible and will play the role conferred upon it, be it principal or secondary. All that is not useful in the picture is detrimental.

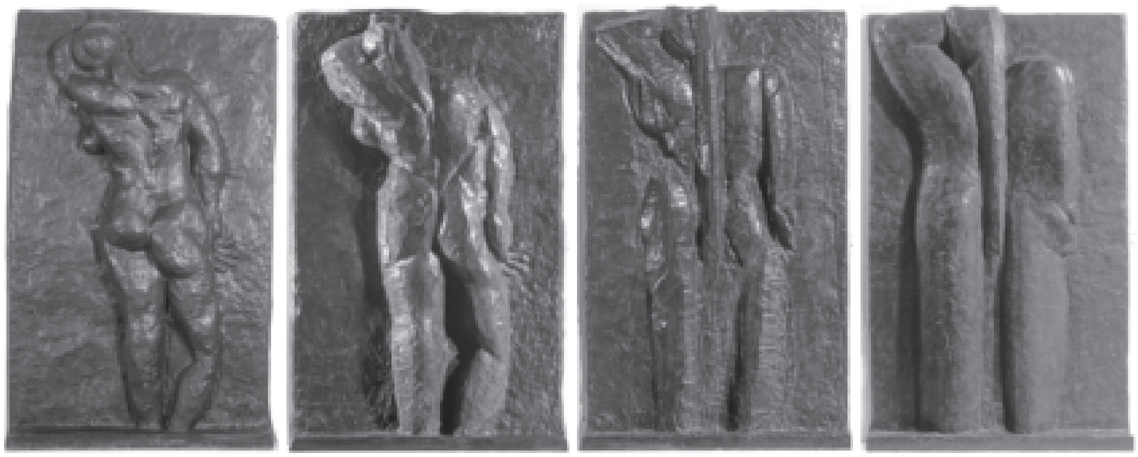


FIGURE 1.2
Bronze sculptures by Henri Matisse. Each rendering is successively more abstract.
Matisse, Henri (1869–1954). *Nude from Behind*. Bas-relief in bronze. © 2015 Succession H. Matisse/Artists Rights Society (ARS), New York; Photo credit: © CNAC/MNAM/Dist. RMN-Grand Palais/Art Resource, NY.

The Back I 1909 | *The Back II* 1913 | *The Back III* 1917 | *The Back IV* 1930

A work of art must be harmonious in its entirety; for superfluous details would, in the mind of the beholder, encroach upon the essential elements.[1]

Suppression of detail is an integral part of the concept of levels of abstraction and carries over directly to computer science. In computer science terminology, *The Back IV* is at the highest level of abstraction and *The Back I* is at the lowest level. FIGURE 1.3 is a level diagram that shows the relationship of these levels.

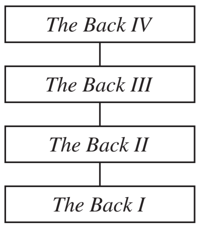


FIGURE 1.3
The levels of abstraction in the Matisse sculptures. *The Back IV* is at the highest level of abstraction.

Like the artist, the computer scientist must appreciate the distinction between the essentials and the details. The chronological progression of creation in *The Back* series was from the most detailed to the most abstract. In computer science, however, the progression for problem solving should be from the most abstract to the most detailed. One goal of this text is to teach you how to think abstractly, that is, to suppress irrelevant detail when formulating a solution to a problem. Not that detail is unimportant in computer science! Detail is most important. However, in computing problems there is a natural tendency to be overly concerned with too much detail in the beginning stages of the progression. In solving problems in computer science, the essentials should come before the details.

1. Alfred H. Barr, Jr., *Matisse: His Art and His Public* (New York: The Museum of Modern Art, 1951).

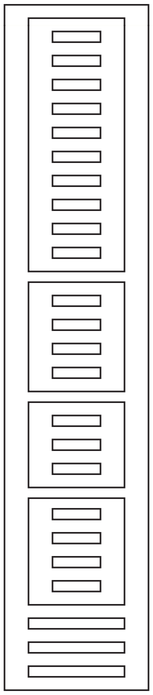
Abstraction in Documents

Levels of abstraction are also evident in the outline organization of written documents. An example is the United States Constitution, which consists of seven articles, each of which is subdivided into sections. The article and section headings shown in the following outline are not part of the Constitution itself.[2] They merely summarize the contents of the divisions.

Article I.	Legislative Department
Section 1.	Congress
Section 2.	House of Representatives
Section 3.	Senate
Section 4.	Elections of Senators and Representatives—Meetings of Congress
Section 5.	Powers and Duties of Each House of Congress
Section 6.	Compensation, Privileges, and Disabilities of Senators and Representatives
Section 7.	Mode of Passing Laws
Section 8.	Powers Granted to Congress
Section 9.	Limitations on Powers Granted to the United States
Section 10.	Powers Prohibited to the States
Article II.	Executive Department
Section 1.	The President
Section 2.	Powers of the President
Section 3.	Duties of the President
Section 4.	Removal of Executive and Civil Officers
Article III.	Judicial Department
Section 1.	Judicial Powers Vested in Federal Courts
Section 2.	Jurisdiction of United States Courts
Section 3.	Treason
Article IV.	The States and the Federal Government
Section 1.	Official Acts of the States
Section 2.	Citizens of the States
Section 3.	New States
Section 4.	Protection of States Guaranteed
Article V.	Amendments
Article VI.	General Provisions
Article VII.	Ratification of the Constitution

2. California State Senate, J. A. Beak, Secretary of the Senate, *Constitution of the State of California, the Constitution of the United States, and Related Documents* (Sacramento, 1967).

FIGURE 1.4
A nesting diagram of the United States Constitution.



The Constitution as a whole is at the highest level of abstraction. A particular article, such as Article III, Judicial Department, deals with part of the whole. A section within that article, Section 2, Jurisdiction of United States Courts, deals with a specific topic and is at the lowest level of abstraction. The outline organizes the topics logically.

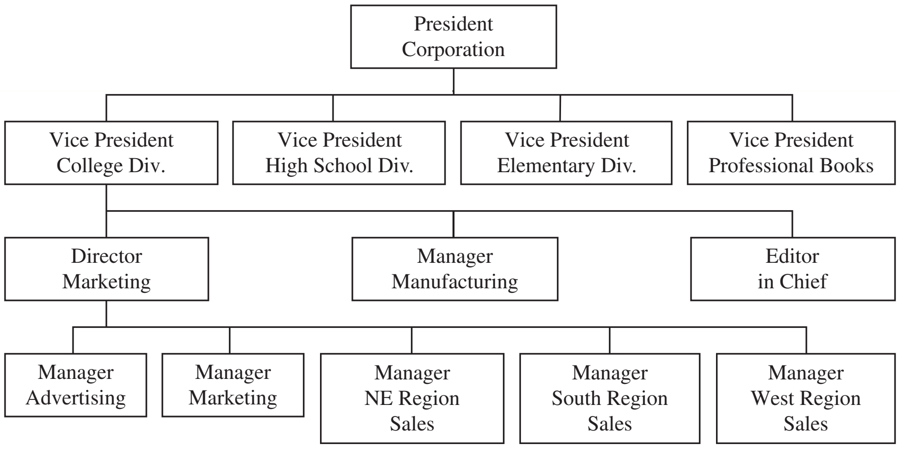
FIGURE 1.4 shows the outline structure of the Constitution in a nesting diagram. The big outer box is the entire Constitution. Nested inside it are seven smaller boxes, which represent the articles. Inside the articles are the section boxes.

This outline method of organizing a document is also important in computer science. The technique of organizing programs and information in outline form is called *structured programming*. In much the same way that English composition teachers instruct you to organize a report in outline form before writing the details, software designers organize their programs in outline form before filling in the programming details.

Abstraction in Organizations

Corporate organization is another area that uses the concept of levels of abstraction. For example, FIGURE 1.5 is a partial organization chart in the form of a hierarchy diagram for a hypothetical textbook publishing company. The president of the company is at the highest level and is responsible for the successful operation of the entire organization. The four vice presidents report to the president. Each vice president is responsible for just one major

FIGURE 1.5
A simplified organization chart for a hypothetical publishing company.


President Corporation
Vice President College Div.
Vice President High School Div.
Vice President Elementary Div.
Vice President Professional Books
Director Marketing
Manager Manufacturing
Editor in Chief
Manager Advertising
Manager Marketing
Manager NE Region Sales
Manager South Region Sales
Manager West Region Sales

part of the operation. There are more levels, not shown in the figure, under each of the managers and vice presidents.

Levels in an organization chart correspond to responsibility and authority in the organization. The president acts in the best interest of the entire company. She delegates responsibility and authority to those who report to her. They in turn use their authority to manage their part of the organization and may delegate responsibilities to their employees. In businesses, the actual power held by individuals may not be directly reflected by their positions on the official chart. FIGURE 1.6 is a level diagram that shows the line of authority in the organization. Each vice president reports to the president. Each director reports to a vice president, each manager reports to a director, and so on down the chain of command.

There is a direct relationship between the way an organization functions, as reflected by its organization chart, and the way a computer system functions. Like a large organization, a large computer system is typically organized as a hierarchy. Any given part of a computer system takes orders from the part immediately above it in the hierarchy diagram. In turn, it issues orders to be carried out by those parts immediately below it in the hierarchy.

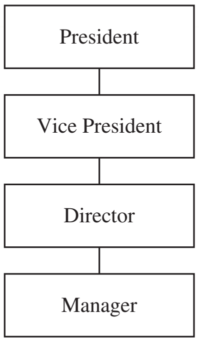


FIGURE 1.6 The level diagram for the organization in Figure 1.5.

Abstraction in Machines

Another example of levels of abstraction that is closely analogous to computer systems is the automobile. Like a computer system, the automobile is a man-made machine. It consists of an engine, a transmission, an electrical system, a cooling system, and a chassis. Each part of an automobile is subdivided. The electrical system has, among other things, a battery, headlights, and a voltage regulator.

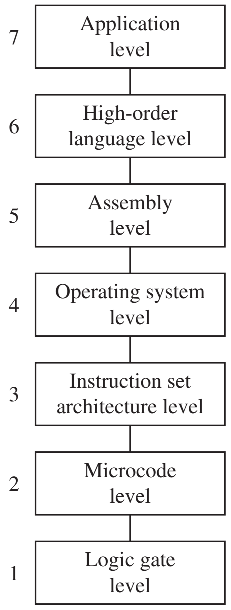
People relate to automobiles at different levels of abstraction. Drivers are at the highest level of abstraction. They perform their tasks by knowing how to operate the car: for example, how to start it, how to use the accelerator, and how to apply the brakes.

At the next lower level of abstraction are the backyard mechanics. They understand more of the details under the hood than the casual drivers do. They know how to change the oil and the spark plugs. They do not need this detailed knowledge to drive the automobile.

At the next lower level of abstraction are the master mechanics. They can completely remove the engine, take it apart, fix it, and put it back together. They do not need this detailed knowledge to simply change the oil.

In a similar vein, people relate to computer systems at many different levels of abstraction. A complete understanding at every level is not necessary to use a computer. You do not need to be a mechanic to drive a car. Similarly, you do not need to be an experienced programmer to use a word processor.

FIGURE 1.7
The level structure of a typical computer system. Some systems do not have Level 2.



Abstraction in Computer Systems

FIGURE 1.7 shows the level structure of a typical computer system. Each of the seven levels shown in the diagram has its own language:

Level 7 (App7):	Language dependent on applications program
Level 6 (HOL6):	Machine-independent programming language
Level 5 (Asmb5):	Assembly language
Level 4 (OS4):	Operating system calls
Level 3 (ISA3):	Machine language
Level 2 (Mc2):	Microinstructions and register transfer
Level 1 (LG1):	Boolean algebra and truth tables

Programs written in these languages instruct the computer to perform certain operations. A program to perform a specific task can be written at any one of the levels of Figure 1.7. As with the automobile, a person writing a program in a language at one level does not need to know the language at any of the lower levels.

When computers were invented, only Levels LG1 and ISA3 were present. A human communicated with these machines by programming them in *machine language* at the instruction set architecture level. Machine language is great for machines but is tedious and inconvenient for a human programmer. *Assembly language*, at Level Asmb5, was invented to help the human programmer.

The first computers were large and expensive. Much time was wasted when one programmer monopolized the computer while the other users waited in line for their turns. Gradually, *operating systems* at Level OS4 were developed so many users could access the computer simultaneously. With today's personal computers, operating systems are still necessary to manage programs and data, even if the system services only one user.

In the early days, every time a company introduced a new computer model, the programmers had to learn the assembly language for that model. All their programs written for the old machine would not work on the new machine. *High-order languages* at Level HOL6 were invented so programs could be transferred from one computer to another with little modification and because programming in a high-order language is easier than programming at a lower level. The more popular Level HOL6 languages that you may be familiar with include the following:

› *C*	For programming operating systems
› *C++*	For general applications; C with added object-oriented features
› *Python*	A scripting language for web applications
› *Java*	For general-purpose and web applications

The widespread availability of computer systems spurred the development of many applications programs, also called *apps*, at Level App7. An *applications program* is one written to solve a specific type of problem, such as composing a document, statistically analyzing data, or routing a car to its destination. It allows you to use the computer as a tool without knowing the operational details at the lower levels.

Level LG1, the lowest level, consists of electrical components called *logic gates*. Along the way in the development toward higher levels, it was discovered that a level just above the logic gate level could be useful in helping designers build the Level ISA3 machine. *Microprogramming* at Level Mc2 is used on some computer systems today to implement the Level ISA3 machine. Level Mc2 was an important tool in the invention of the handheld calculator.

One goal of this text

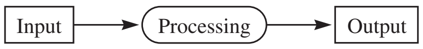
Your goal in studying this text is to communicate effectively with computers. To do so, you must learn the language. Languages at the higher levels are more human-oriented and easier to understand than languages at the lower levels. That is precisely why they were invented.

As you study this text, you will gain some insight into the inner workings of a computer system by examining successively lower levels of abstraction. The lower you go in the hierarchy, the more details will come to light that were hidden at the higher levels. As you progress in your study, keep Figure 1.7 in mind. You must master a host of seemingly trivial details; it is the nature of the beast. Remember, however, that the beauty of computer science lies not in the diversity of its details but in the unity of its concepts.

1.2 Hardware

We build computers to solve problems. Early computers solved mathematical and engineering problems, and later computers emphasized information processing for business applications. Today, computers also control machines as diverse as automobile engines, robots, and microwave ovens. A computer system solves a problem from any of these domains by accepting input, processing it, and producing output. FIGURE 1.8 illustrates the function of a computer system.

FIGURE 1.8
The three activities of a computer system.



Computer systems consist of hardware and software. *Hardware* is the physical part of the system. Once designed, hardware is difficult and expensive to change. *Software* is the set of programs that instruct the hardware and is easier to modify than hardware. Computers are valuable because they are general-purpose machines that can solve many different kinds of problems, as opposed to special-purpose machines that can each solve only one kind of problem. Different problems can be solved with the same hardware by supplying the system with a different set of instructions—that is, with different software.

Every computer has three basic hardware components:

Components of hardware

- Central processing unit (CPU)
- Main memory
- Disk

FIGURE 1.9 shows these components in a block diagram. The lines between the blocks represent the flow of information. The information flows from one component to another on the *system bus*, which is simply a group of wires connecting the components. The preceding list of hardware components is in order of increasing storage capacity. The CPU has the smallest storage capacity, and the disk has the largest capacity. The list is also in order of decreasing speed. The CPU is the fastest device, and the disk is the slowest device.

Central Processing Unit

Registers are in the CPU.

Processing occurs in the CPU, which contains the circuitry to control all the other parts of the computer. It has a small set of memory, called *registers*, represented by the two blocks inside the CPU block in Figure 1.9. Although the figure shows only two registers, they typically number anywhere from

FIGURE 1.9
Block diagram of the three components of a computer system.

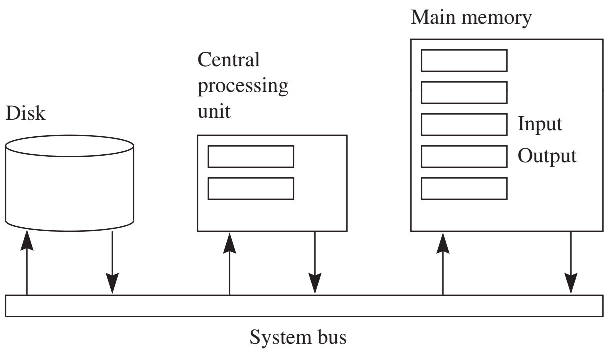
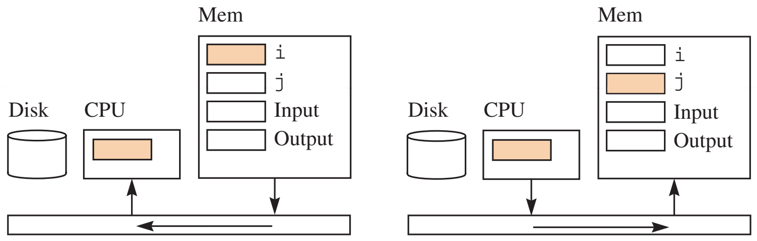


FIGURE 1.10
Information flow to execute the Level HOL6 statement `j = i + 1`.



(a) The first Level ISA3 instruction: fetch the value of `i`.

(b) The third Level ISA3 instruction: store the sum to `j`.

16 to 64. The CPU also has a set of instructions permanently wired into its circuitry. The instructions do such things as fetch information from main memory into a register; add, subtract, compare, and store information from a register back into main memory; and so on. What is not permanent is the order in which these instructions are executed. The order is determined by a program written in machine language at Level ISA3.

An example of how the CPU processes information is execution of the simple assignment `j=i+1`, which is a programming statement in Level HOL6 languages like C or Java. It adds 1 to integer variable `i` and assigns the sum to `j`. During execution of a program, the system stores the values of variables `i` and `j` in main memory. The single `j=i+1` statement at Level HOL6 is translated to three statements at Level ISA3. Each of these three statements is one of the instructions wired into the CPU. The first instruction fetches the current value of `i` from memory into a CPU register, as FIGURE 1.10(a) shows. Information flows from the storage cell in memory onto the system bus and into a register in the CPU. The second instruction adds 1 to the value in the register. This instruction executes entirely within the CPU and involves no information flow along the system bus. The third instruction stores the incremented value back to the location of `j` in the main memory, as Figure 1.10(b) shows.

One statement at Level HOL6 can be three statements at Level ISA3.

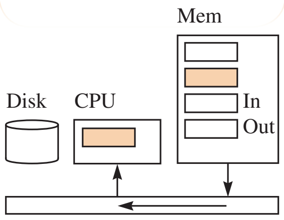
Main Memory

Like the CPU, *main memory* has a set of memory cells to store information. It differs in two respects from the CPU. First, the number of storage cells is far greater than the number of cells in the CPU. Figure 1.9 shows only five cells in main memory, but a smartphone can have a few billion cells and a laptop more than 10 billion. These numbers compare with only dozens of registers in a typical CPU. Second, the CPU has a set of instructions wired into its circuitry to perform processing on the data in its registers. Although the capacity of main memory is huge compared to the capacity of the CPU, it cannot process the values that it stores. The only function it can perform is to remember the data values stored in its cells and produce them for the CPU or the disk on request.

Main memory stores four kinds of information:

- Data to be processed by the CPU
- Program instructions to be executed by the CPU
- Input connections to receive data from the external environment
- Output connections to send data to the external environment

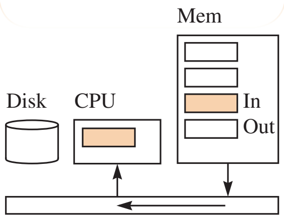
FIGURE 1.11 Information flow to fetch an instruction to be executed.



An example of data storage are the integer values for `i` and `j` in Figure 1.10. Later during execution, if the program needs the value of `j` for another computation, main memory will deliver the value unchanged from the value previously stored.

When you purchase an application, say a word processor, you download it to the disk, which stores it until you want to use it. Then, when you execute the app, your computer system sends a copy of it from the disk to main memory. At any given time, main memory contains copies of all apps that are executing in the system. So, in addition to the values of variables `i` and `j` in Figure 1.10, main memory also stores the program instructions of the app. Because the CPU contains the circuitry to execute the app instructions, the computer system must fetch an instruction from main memory before it can execute the instruction. FIGURE 1.11 shows the information flow to fetch an instruction. The information flow is similar to that in Figure 1.10(a) across the system bus, except that the information comes from the location where the instruction is stored instead of where `i` is stored. The instruction is copied to a special-purpose register in the CPU called the *instruction register*. The electronic circuitry in the CPU is designed to analyze the instruction in the instruction register and to execute it.

FIGURE 1.12 Information flow to receive a character from the keyboard.



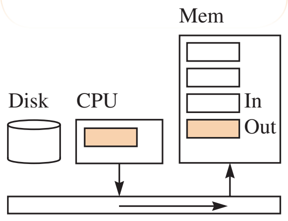
Memory-mapped I/O

A few locations in main memory are reserved for input connections and output connections. This technique is called *memory-mapped input/output*, or *memory-mapped I/O*. A common example of an input device mapped to main memory is the keyboard. When you press a key on the keyboard, the circuitry inside the keyboard detects which key was pressed and sends information representing the character to the input connection in main memory. Then the keyboard sends a signal to the CPU informing it that a key was pressed. In response to the notification signal, the CPU fetches the character from the input connection so it can be processed. FIGURE 1.12 shows the information flow to receive a character from the keyboard. In the same way that information flows when fetching data or an instruction from memory, the input information flows along the system bus to the CPU. The only difference is that the information comes from the input connection of the keyboard that is wired into the storage cell in main memory.

Figure 1.12 might give the impression that the computer system has one large main memory device and that the keyboard is physically connected to one of the storage cells of that single device. Such a configuration is a convenient model for the programmer at Level ISA3. To get data from an input device, the ISA3 programmer writes an instruction to fetch the data from the input connection using the same coding techniques he would use to get the data value from a variable. The hardware designer at Level LG1, however, uses several separate devices to construct the main memory of the system. The designer does not connect the keyboard physically to a storage cell in a single central memory device. Instead, she connects the keyboard to the system bus to make it appear to the ISA3 programmer that there is a single memory device. This abstraction hides the connection details at Level LG1 from the programmer at Level ISA3.

The mouse, the trackpad, and the touch screen are three other common input devices. Inside a mouse is a small light-emitting diode that shines a beam of light down onto the surface of the desk. The light reflects back onto a sensor that samples the light 1500 times per second. A digital signal processor inside the mouse acts like a tiny computer programmed for only one task: to detect patterns in the images of the desktop and determine how far they have moved since the previous sample. The processor inside the mouse computes the direction and velocity of the mouse from the patterns and sends the data through the input connection to the computer, which in turn draws the cursor image on the screen.

FIGURE 1.13
Information flow to send data to the output connection.



The most visible output device is the screen of the computer system. Other devices include speakers for audio output and printers for hard copy output. In a small system like a handheld calculator, main memory contains one output connection for each number or letter that is visible on the calculator display. After the calculator computes the number to display, it sends each character of the number to the corresponding output connection. For example, if the number to display is 263, the computer sends the character `2` to the first output connection, the character `6` to the second output connection, and the character `3` to the third output connection. FIGURE 1.13 shows the information flow over the system bus for one of the characters in the number.

Pixels

Tablets and personal computers have pixel displays. A *pixel* is a picture element, which is a single dot on the display. The system forms an image on the display by setting the brightness and color of each individual pixel. Printers also form images by coloring each pixel on the paper with the proper mixture of ink. It is theoretically possible to have an output connection in main memory for each pixel in the output device. However, a typical screen can have around 10 million pixels. If the CPU had the task of keeping each pixel updated with the proper value, it would not have enough time to perform any other computations.

Consequently, the main memory output connections for displays are not attached directly to each pixel of the display. Rather, they are attached to I/O modules containing separate special-purpose processors whose only function is to compute the color values for each pixel of the display. For example, to draw a line on the display, the CPU does not need to calculate the color values for all the pixels to render the line. Instead, it can send the endpoints of the line to the output connections, and then the I/O module can use that information to calculate and set the color values of the pixels in the display.

Disk

Like the CPU and main memory, the *disk* has a set of memory cells to store information. It differs from main memory in three respects. First, main memory is *volatile*. That is, when you turn the power off or when there is a power outage, the values in main memory are lost. In contrast, disk memory is nonvolatile. You can store information on a disk, turn the power off, and when you turn the power back on, the information is still there.

Disk memory is nonvolatile.

Second, the storage capacity of a disk is about a thousand times the capacity of main memory. The number of storage cells in main memory is typically in the billions, but the number of cells on a disk is typically in the trillions.

Disk memory capacity is greater than main memory capacity.

Third, although disk has high storage capacity, it has low speed compared to main memory. This speed difference is due to the different access methods of the two types of storage. The access method for main memory is *random*. In fact, the electronic components that make up main memory are often referred to as RAM (for *random access memory*) circuits. If you have just fetched some information from one end of main memory, you can immediately get information from the other end at random without passing over the information in between.

Random access

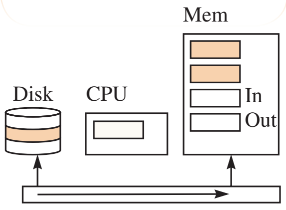
In contrast, the access method for a disk is *sequential*. A hard disk drive consists of a spinning platter with a thin magnetic coating on its surface that stores the information. A tiny sensor skims the surface of the disk and can move from the center to the edge of the disk to locate the position on the surface where the information is stored. If you have just fetched some information near the center of the disk, you cannot get information from the edge of the disk until you first move the sensor to the edge and then wait for the information to rotate to the sensor. The time to move the sensor to the proper location is called the *seek time*, and the time to wait for the information to rotate under the sensor is called the *latency*. These times are typically a few thousandths of a second. The time to access a value from RAM is typically a few billionths of a second. So, disk is about a million times slower than main memory.

Sequential access

Disk access is slower than main memory access.

Solid-state disks (SSDs) perform the same function as hard disk drives but are all electronic, with no moving parts. They do not have the high storage capacity of hard disk drives, but they are about a hundred times faster because they have no moving parts. The lack of moving parts makes them more reliable as well. Still, even an SSD is thousands of times slower than main memory.

FIGURE 1.14 Direct memory access between a disk and main memory.



Direct memory access

FIGURE 1.14 shows the information flow between disk and main memory over the system bus. It differs from the previous information flows because the CPU is not an intermediary in the transaction. Information transfer between disk and main memory without going through the CPU registers is called *direct memory access,* or DMA. The disk has its own special-purpose processor, called a *DMA controller*, whose only function is to transfer information over the system bus between the disk and main memory. The CPU initiates the transfer request by sending a signal to the DMA controller. Then, while the CPU processes the information in its registers, the DMA controller simultaneously transfers the information over the bus. The disk controller signals the CPU when the transfer is complete.

Suppose you have a document you need to edit in a word processor. The word processor app is stored on disk. When you start up the app, the system sends a copy of the app from the disk to memory with a DMA transfer, as in Figure 1.14. Then, when you open your document, the system does the same kind of transfer, bringing your document into main memory. Figures 1.12 and 1.13 show the information flow when you type a key on the keyboard and view the results on the screen. When you save your document, the system does a DMA transfer similar to the one of Figure 1.14 but in the other direction, transferring your document from main memory to disk.

1.3 Software

Algorithms

An *algorithm* is a set of instructions that, when carried out in the proper sequence, solves a problem in a finite amount of time. Algorithms do not require computers. FIGURE 1.15 is an algorithm in English that solves the problem of making six servings of stirred custard.

This recipe illustrates two important properties of algorithms—the finite number of instructions and execution in a finite amount of time. The algorithm has seven instructions—combine, stir, cook, remove, cool, add, and chill. Seven is a finite number. An algorithm cannot have an infinite number of instructions.

Even though the number of instructions in the custard algorithm is finite, there is a potential problem with its execution. The recipe instructs us to cook until the custard coats the metal spoon. What if it never coats the

FIGURE 1.15
An algorithm for making stirred custard.

Ingredients
- 3 slightly beaten eggs
- ¼ cup sugar
- 2 cups milk, scalded
- ½ teaspoon vanilla

Algorithm
- Combine eggs, sugar, and ¼ teaspoon salt.
- Slowly stir in slightly cooled milk.
- Cook in double boiler over hot, not boiling, water, stirring constantly.
- As soon as custard coats metal spoon, remove from heat.
- Cool at once—place pan in cold water and stir a minute or two.
- Add vanilla.
- Chill.

spoon? Then, if we strictly followed the instructions, we would be cooking forever! A valid algorithm must never execute endlessly. It must provide a solution in a finite amount of time. Assuming that the custard will always coat the spoon, this recipe is indeed an algorithm.

The finite requirement for an algorithm

Definition of a program

A *program* is an algorithm written for execution on a computer. Programs cannot be written in English. They must be written in a language for one of the seven levels of a computer system.

General-purpose computers can solve many different kinds of problems, from computing the company payroll to correcting a spelling mistake in a memorandum. The hardware gets its versatility from its ability to be programmed to do the different jobs. Programs that control the computer are called *software*.

Software

Software is classified into two broad groups:

- Systems software
- Applications software

Systems software versus applications software

Systems software makes the computer accessible to the applications designers. *Applications software*, in turn, makes the computer system accessible to the end user at Level App7. Generally speaking, a systems software engineer designs programs at Level HOL6 and below. These programs take care of the many details of the computer system with which the applications programmer does not want to bother.

Operating Systems

The most important software for a computer is the operating system. The *operating system* is the systems program that makes the hardware usable. Every general-purpose computer system includes both hardware and an operating system.

To study this text effectively, you must have access to a computer with an operating system. Some common commercial operating systems are Microsoft Windows, Mac OS X, Linux, Android, and iOS. Mac OS X and iOS are Unix operating systems, and Android is based on Linux, which is in turn a version of Unix.

An operating system has three general functions:

Functions of an operating system

- File management
- Memory management
- Processor management

Of these three functions, file management is the most visible to the user. The first thing a new computer user learns is how to manipulate the files of information on the operating system.

Disks, directories, and files in an operating system are analogous to filing cabinets, file folders, and documents in an office. In an office, the filing cabinet stores file folders, each of which contains a set of documents. In an operating system, disk drives store directories, each of which contains a set of files. The office worker assigns a name to a file folder that reflects the contents of the folder and makes it easy to pick out an individual file from the cabinet. Similarly, the computer user assigns a name to a directory that reflects the contents of the directory and makes it easy to pick out a file to open in an application.

Files can contain three types of information:

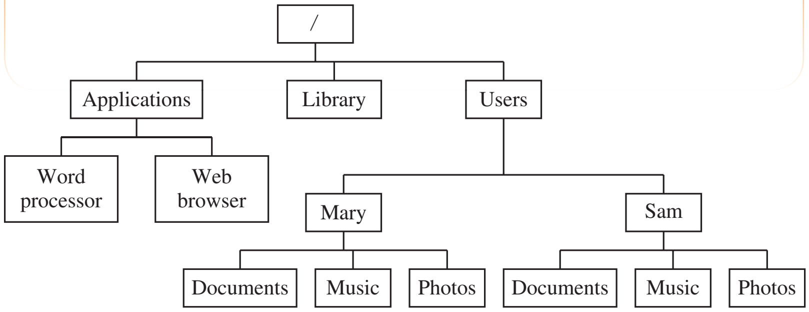
Three types of information contained in files

- Documents
- Programs
- Data

Documents may be company memoranda, music, photos, and the like. Files also store programs to be executed by the computer. To be executed, first they must be loaded from disk into main memory, as Figure 1.14 shows. Input data for an executing program can come from a file, and output data can also be sent to a file.

The files are physically scattered over the surface of the disk. To keep track of all these files of information, the operating system maintains a

FIGURE 1.16
A typical hierarchical file system.



directory of them. The directory is a list of all the files on the disk. Each entry in the directory has the file's name, its size, its physical location on the disk, and any other information the operating system needs to manage the files. The directory itself is also stored on the disk.

FIGURE 1.16 shows a typical file system, which is hierarchical from the user's perspective. The box at the top is the root directory, labeled / in Unix systems. Below it are directories for storing applications, a library of software used by the applications, and each user's login account for the computer. Each user has a separate directory for files of documents, music, and photos.

The operating system provides the user with a way to manipulate the files on the disk. Common operating system commands include changing the name of a file or directory, deleting a file from the disk, and executing an application program. Experienced programmers can execute such commands on the command line with a terminal app. Most users initiate the commands with a point and click of the mouse. Your operating system is a program written for your computer by a team of systems programmers. When you issue the command to delete a file from the disk, a systems program executes that command. You, the user, are using a program that someone else, the systems programmer, wrote.

Software Analysis and Design

Software, whether systems or applications, has much in common with literature. Human authors write both. Other people read both, although computers can also read and execute programs. Both novelists and programmers are creative in that the solutions they propose are not unique. When a novelist has something to communicate, there is always more than one way to express it. The difference between a good novel and a bad one lies not only in the idea communicated, but also in the way the idea is expressed. Likewise, when a programmer has a problem to solve, there is always more than one way to program the solution. The difference between a good program and a bad one lies not only in the correctness of the solution to the problem, but also in other characteristics of the program, such as clarity, execution speed, and memory requirement.

As a student of literature, you participate in two distinct activities—reading and writing. Reading is analysis; you read what someone else has written and analyze its contents. Writing is design or synthesis; you have an idea to express, and your problem is how to communicate that idea effectively. Most people find writing more difficult than reading, because it requires more creativity. That is why there are more readers in the general population than authors.

Similarly, as a student of software, you will analyze and design programs. Remember that the three activities of a program are input, processing, and output. In analysis, you are given the input and the processing instructions. Your problem is to determine the output. In design, you are given the input and the desired output. Your problem is to write the processing instructions, that is, to design the software. FIGURE 1.17 shows the difference between analysis and design.

Analysis versus design

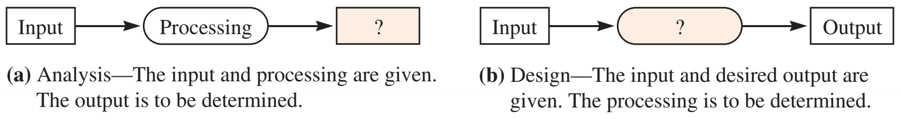
As in reading and writing English literature, designing good software is more difficult than analyzing it. A familiar complaint of computer science students is "I understand the concepts, but I can't write the programs." This is a natural complaint because it reflects the difficulty of synthesis as opposed to analysis. Our ultimate goal is for you to be able to design software as well as analyze it. The following chapters will give you specific software design techniques.

But first you should become familiar with these general problem-solving guidelines, which also apply to software design:

General problem-solving guidelines

- Understand the problem.
- Outline a solution.
- Solve each part of your outlined problem.
- Test your solution on the computer.

FIGURE 1.17
The difference between analysis and design.



(a) Analysis—The input and processing are given. The output is to be determined.

(b) Design—The input and desired output are given. The processing is to be determined.

When faced with a software design problem, test your understanding of the problem by writing down some sample input and the corresponding output. You cannot solve a problem by computer if you do not know how to solve it by hand. To outline a solution, you must break down the problem into several subproblems. Because the subproblems are smaller than the original problem, they are easier to solve.

FIGURE 1.18 The eight values possible with three bits.

Decimal	Binary
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

1.4 Digital Information

We live in a space–time universe. Every event in the universe occurs at a particular point in space and at a particular point in time. All computations in the universe, therefore, also occupy space and time. The space in a computer system consists of the electronic devices on the chip circuitry and the wires connecting them. The elapsed time of a computation consists of the time for the CPU to execute the instructions of the program plus the time to move the information between the components of the system.

Quantifying Space

Definition of bit

Because computers are electronic, they store and process information in the form of electrical signals. The signals are voltage levels inside the electronic circuits. Each signal is either at a high voltage level, represented by the digit 1, or a low level, represented by the digit 0. Because there are only two possible values for the signal, it is called a *binary digit* or *bit*. A bit is the smallest unit of digital information. Each bit in a computer system occupies space in the circuit comprised of the electronic components that maintain its value.

Most data values represent either numbers or text. To process such data, a computer system represents numbers and text as a sequence of bits. The number of bits in a sequence is to a certain extent arbitrary. For integers, a long sequence of bits can store a wider range of numeric values than a small sequence can. For text, a long sequence of bits can store a wider range of characters. Different parts of a computer system have bit sequences with different lengths. For example, the registers in the CPU are usually sequences of 32 or 64 bits. The memory cells in main memory are always a sequence of 8 bits.

Because each bit can have two values, either 0 or 1, the number of values that can be stored in a sequence of n bits is fixed as follows:

Information capacity of an n-bit cell

› The number of values stored by a sequence of n bits is 2^n.

For example, FIGURE 1.18 shows all the possible values that can be stored by a sequence of three bits. Because $2^3 = 8$, there are eight possible values that a sequence of three bits can store. If the three bits represent an integer, the eight integer values are 0, 1, 2, . . . , 7, as written in decimal.

How many bits does it take to store a single character in the English language? There are 26 letters in the alphabet. To include both uppercase letters and lowercase letters would require 2 × 26, which is 52 values. To include the 10 digit characters `0` through `9` brings the number of values up to 62. To include all the punctuation marks, like `?` and `!`, add another dozen or so, which brings the total up to about 74. Now, six bits is not enough because 2^6 is 64 and we need to store at least 74 values. Seven bits works because 2^7 is 128, which exceeds our requirement of storing 74 values. The American Standard Code for Information Interchange (ASCII) specifies all possible 128 binary values and the English character that each bit pattern represents. It is common for storing textual information in computer systems. For example, a computer stores the lowercase letter `q` as the sequence of seven bits, 1110001, and the lowercase letter `r` as the sequence of seven bits, 1110010. See the back cover of this book for a table of ASCII characters.

Definition of byte

A sequence of eight bits is called a *byte*, pronounced "bite." On all computer systems, each memory cell in main memory is one byte. To make a single seven-bit character fit into an eight-bit memory cell, the system prefixes an extra zero at the beginning of the seven-bit code for the character. So, `q` is stored as 0111 0001, and `r` is stored as 0111 0010. The rule of thumb to remember is:

Storage requirement for an ASCII character

- It takes one byte, which is eight bits, to store one ASCII character.

FIGURE 1.19 shows the common metric prefixes for representing small quantities. An example of using a prefix for a small quantity would be to quote the seek time for a hard drive as 12 ms. The figure shows that the prefix letter *m* is an abbreviation for *milli-*, so the seek time is 12 milliseconds, or 12×10^{-3} seconds, or 0.012 seconds. Similarly, if the access time of a memory device is 430 ns, that time is 430×10^{-9} seconds, or 0.00000043 seconds.

FIGURE 1.19
Prefixes for small numbers in scientific notation.

Multiple	Prefix	Prefix Letter
10^{-3}	milli-	m
10^{-6}	micro-	μ
10^{-9}	nano-	n
10^{-12}	pico-	p

FIGURE 1.20
Prefixes for large numbers in scientific notation.

Decimal Multiples	Decimal Prefix	Decimal Prefix Letter	Binary Multiples	Binary Prefix	Binary Prefix Letters
$10^3 = 1000$	kilo-	K	$2^{10} = 1024$	kibi-	Ki
$10^6 = 1000^2$	mega-	M	$2^{20} = 1024^2$	mebi-	Mi
$10^9 = 1000^3$	giga-	G	$2^{30} = 1024^3$	gibi-	Gi
$10^{12} = 1000^4$	tera-	T	$2^{40} = 1024^4$	tebi-	Ti
$10^{15} = 1000^5$	peta-	P	$2^{50} = 1024^5$	pebi-	Pi

(a) The decimal and binary prefixes.

Decimal Multiples	Binary Multiples	Percent Difference
$10^3 = 1000$	$2^{10} = 1024$	2.4%
$10^6 = 1,000,000$	$2^{20} = 1,048,576$	4.9%
$10^9 = 1,000,000,000$	$2^{30} = 1,073,741,824$	7.4%
$10^{12} = 1,000,000,000,000$	$2^{40} = 1,099,511,627,776$	10.0%
$10^{15} = 1,000,000,000,000,000$	$2^{50} = 1,125,899,906,842,624$	12.6%

(b) The differences between the decimal and binary values.

The decimal prefixes in FIGURE 1.20(a) for the large values are common in science. For example, 45 MW is 45 megawatts, or 45×10^6 watts. These prefixes are also common for specifying hard disk drive capacities. However, they are not so common for specifying other memory capacities because the access methods for those devices are binary. Consequently, counting is frequently performed in base 2 instead of base 10. To distinguish between the two counting bases, the binary prefixes are modified by including the lowercase letter *i* after the corresponding decimal prefix letter. The binary prefixes in the figure show that a *binary kilo-* is designated as a *kibi-* and is 2^{10}, or 1024, and a *binary mega-* is 2^{20}, or 1024^2. Figure 1.20(b) shows the exact counting values for the decimal and corresponding binary multiples. For the first multiple, the difference between 1000 and 1024 is less than 3%, so you can think of a binary kilo-, or a kibi-, as being about 1000, even though it is a little more. The approximation for a mega- is a little worse but is still within 5%. The same applies to giga-, tera-, and peta-, but for peta- the difference is about 12.6%.

Abbreviations for byte and bit

The abbreviation for a byte is the uppercase letter *B*. The abbreviation for a bit is the lowercase letter *b*. So, for example, a hard disk drive with a capacity of 780 GB can store 780×10^9 bytes using the standard meaning of giga-. On the other hand, a main memory of 8 GiB can store 8×2^{30} bytes, which is $8 \times 1{,}073{,}741{,}824$ bytes, or 8.59×10^9 bytes. As of this writing, the adoption of the binary prefixes is far from complete in the computing industry. Before the binary prefixes were standardized, the decimal prefixes were used to refer to both decimal values and binary values with no distinction between the two notations. Some manufacturers still quote main memory capacity, for example, as *8 GB* when they mean *8 GiB*. As another example, many computer engineers still refer to a *64-KB* storage unit when they mean *64 KiB*.

Disk drive

Quantifying Time

All computations in a computer system occupy time as well as space. The two components of time in a computer system are the computation time, which is the time it takes the CPU to execute one of the instructions in its instruction set, and the transmission time, which is the time it takes to move information from one component of the system to another. The binary prefixes in Figure 1.20(a) are never used for specifying time.

A single machine instruction is fast by human standards. CPU speeds are commonly measured in GHz, which stands for *gigahertz*, a unit of frequency. A hertz is one cycle per second. So, a GHz is a billion cycles per second. For example, a CPU rated at 4.6 GHz executes at a rate of 4.6 billion cycles per second. Each computer instruction at Level Mc2 requires one cycle to execute. Each instruction at Level ISA3, however, is composed of several instructions at Level Mc2. When you purchase an app, you get a program of Level ISA3 instructions. So, when an ISA3 instruction in your app executes, it requires the execution of several Mc2 instructions, each of which requires one cycle.

Suppose you are running an app and you click on a button to perform a task. The following system performance equation computes the total execution time of the program task as the product of three terms:

The system performance equation

$$\frac{\text{time}}{\text{program}} = \frac{\text{instructions}}{\text{program}} \times \frac{\text{cycles}}{\text{instruction}} \times \frac{\text{time}}{\text{cycle}}$$

The first term is the number of Level ISA3 instructions that execute to perform the task. The second term is the average number of Mc2 instructions it takes to execute a single ISA3 instruction. The third term is the time it takes to complete one cycle and is related to the frequency as

$$T = \frac{1}{f}$$

where T, the period, is the number of seconds per cycle and f, the frequency rating of the CPU, is the number of cycles per second. As in all scientific calculations, the units in the three terms cancel to give the units of the product. *Instructions* in the numerator of the first term cancels with *instruction* in the denominator of the second term, and *cycles* in the numerator of the first term cancels with *cycle* in the denominator of the third term, to yield time per program for the result.

The number of cycles per ISA3 instruction varies greatly. Some ISA3 instructions are composed of only a single Mc2 cycle. Examples are instructions to add or subtract two integers and instructions to move a value from one register in the CPU to another. Some are composed of a few cycles. For example, to multiply two integers can take about four or five cycles. A complex computation like taking the cosine of a double value can take about 100 cycles. The average number of cycles per instruction in a given program task depends on the mix of ISA3 instructions that execute to perform the task.

Example 1.1 Suppose your CPU is rated at 2.5 GHz and you execute a program task on your app that requires the execution of 16 million ISA3 instructions. If each ISA3 instruction executes an average of 3.7 Mc2 instructions, what is the execution time of the program task?

$$\begin{aligned}
& \text{time/program} \\
= \quad & \langle \text{System performance equation} \rangle \\
& (\text{instructions/program}) \times (\text{cycles/instruction}) \times (\text{time/cycle}) \\
= \quad & \langle \text{Substitute values with } T = 1/f \text{ for the third term} \rangle \\
& (16 \times 10^6) \times (3.7) \times (1/(2.5 \times 10^9)) \\
= \quad & \langle \text{Math} \rangle \\
& 23.7 \text{ ms}
\end{aligned}$$

So, the time is 23.7×10^{-3} seconds, or about 0.024 seconds. ■

This example shows how to calculate the time it takes to execute a program task. The other aspect of time in a computer system is the time to move information from one component of the system to another. In general, the connection between the source and the destination over which information flows is called a *channel*. A channel could be the wires comprising the system bus in a computer system, or it could be the space between a cell phone tower and a cell phone. The *bandwidth* of the channel is the quantity of information that it can carry per unit of time. Channels with a high bandwidth can transfer more information per second than channels with a low bandwidth. It is common to quote channel bandwidths in units of bits per second, although occasionally it is quoted in bytes per second. Because a byte is eight bits, a channel with a bandwidth of 8.2 MB/s, for example, is equivalent to 8×8.2 or 65.6 Mb/s.

The bandwidth of a channel

The following bandwidth equation computes the total information transferred as the product of two terms:

$$\text{information} = \frac{\text{information}}{\text{time}} \times \text{time}$$

The bandwidth equation

The first term in the product is the bandwidth of the channel, and the second term is the transmission time.

Example 1.2 The bandwidth of a DMA channel between the hard drive and main memory in a computer system is quoted as 3 Gb/s. How long would it take to do a DMA transfer from the hard drive to main memory of the 400-MB thumbnail database from the photo library in your computer?

$$\begin{aligned}
&\quad \text{time} \\
&= \quad \langle \text{Solve bandwidth equation for time} \rangle \\
&\quad \text{information}/(\text{information}/\text{time}) \\
&= \quad \langle \text{Substitute values with 8 bits per byte for information} \rangle \\
&\quad 8 \times 500 \times 10^6/3 \times 10^9 \\
&= \quad \langle \text{Math} \rangle \\
&\quad 1.33 \text{ s}
\end{aligned}$$

▮

Example 1.3 A typist is entering some text on a computer keyboard at the rate of 35 words per minute. How large must the bandwidth of the channel be to accommodate the information flow between the typist and the computer system? Assume that each word is followed by one space character, on average.

Including one space character after each word, the typist enters 36 characters per minute.

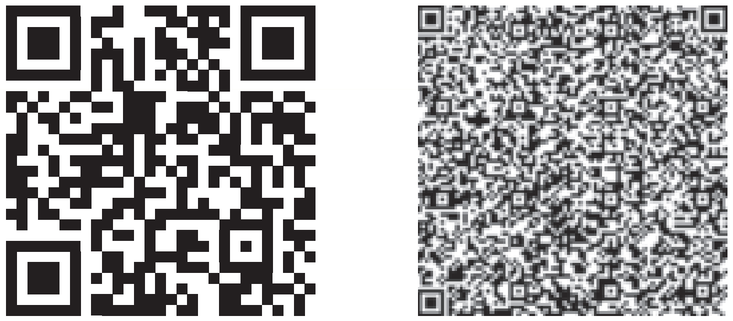
$$\begin{aligned}
&\quad \text{bandwidth} \\
&= \quad \langle \text{Definition of bandwidth} \rangle \\
&\quad \text{information}/\text{time} \\
&= \quad \langle \text{Substitute values} \rangle \\
&\quad (8(\text{b/char}) \times 36 \text{ char})/(1(\text{min}) \times 60(\text{s/min})) \\
&= \quad \langle \text{Math} \rangle \\
&\quad 4.8 \text{ b/s}
\end{aligned}$$

▮

Quick Response Codes

The quick response code, or *QR code*, was invented in Japan to track items in the automotive industry. It is so versatile that it is now used for storing all kinds of textual information that can be conveniently scanned on mobile devices with built-in cameras. FIGURE 1.21 shows two QR codes. Figure 1.21(a) shows the code for a web URL, and Figure 1.21(b) shows the code for the text of the first paragraph of this chapter.

FIGURE 1.21
Two QR codes.



(a) The QR code for a web URL.

(b) The QR code for the text of the first paragraph of this chapter.

A QR code is a square matrix of light and dark boxes of any color, each of which stores one bit of information. The QR terminology for a single box is a *module*. The following discussion refers to a module as a bit that stores 1 if it is black and 0 if it is white. There are 40 versions of QR codes, ranging in size from 21 × 21 bits for Version 1, 25 × 25 bits for Version 2, up to 177 × 177 bits for Version 40. Each version adds four bits per dimension to the previous version. Figure 1.21(a) is Version 3 with 29 × 29 bits, and Figure 1.21(b) is Version 11 with 81 × 81 bits. The larger the version, the more information can be stored. The code in part (b) stores all 564 characters of the first paragraph of this chapter.

Seven parts of the grid are reserved for alignment and format purposes as follows:

The QR code alignment and format regions

- Finder patterns
- Separators
- The dark module
- The format information area
- Timing patterns
- Alignment patterns (for Version 2 and higher only)
- The version information area (for Version 7 and higher only)

FIGURE 1.22(a) shows the first six of these regions for the Version 3 QR code of Figure 1.21(a). This version has no reserved version information area.

The finder patterns

The finder patterns located on three corners of the code are 7- × 7-square blocks that help the scanner determine the orientation of the code. Each consists of a black border seven bits wide surrounding a white border five bits wide, which in turn surrounds a 3 × 3 black matrix. The total number of bits occupied by the finder patterns is 7 × 7 × 3, or 147 bits. The separators are the white borders around the inside boundaries of the finder patterns and occupy 15 bits each, for a total of 45 bits for the three. The dark module is a single black bit at the upper-right corner of the separator for the lower-left finder pattern. The format information area along the sides of the separators is shaded in the figure and totals 30 bits. These first four reserved regions always occupy the same area on the code and total 147 + 45 + 1 + 30, which is 223 bits.

The separators

The dark module

The format information area

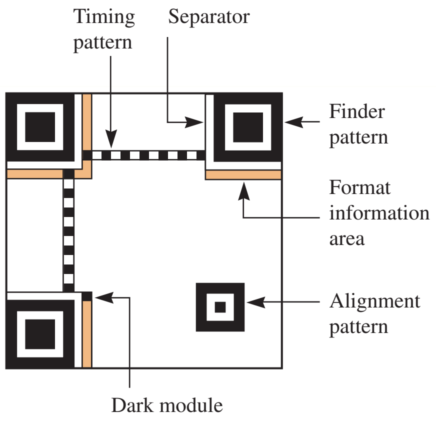
The timing patterns help the scanner identify the individual rows and columns in the grid. Each is a single track of alternating ones and zeros. The number of bits occupied by a timing pattern depends on the version of the code. The higher the version, the more bits in the timing pattern. For Version 3 with a 29- × 29-bit grid, the length is 29 minus the 8 bits on each end occupied by the finder pattern and separator. That is 13 bits for each timing pattern and 26 bits total for the two patterns.

The timing patterns

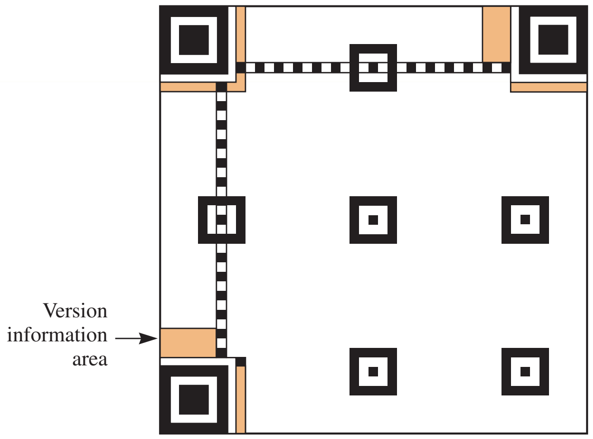
Each alignment pattern is a 5- × 5-square block and hence occupies 25 bits. However, the number of alignment patterns varies with the version, and the placement is such that an alignment pattern can intersect with a timing pattern. Version 3 in Figure 1.22(a) has one alignment pattern, which does not intersect a timing pattern. You should be able to spot the 13 alignment patterns in the QR code of Figure 1.21(b). Two of them intersect the top timing pattern, and two of them intersect the left timing pattern. To

The alignment patterns

FIGURE 1.22
The reserved regions in a QR code.



(a) The regions in the Version 3 QR code of Figure 1.21(a).



(b) The regions in a Version 7 code.

calculate the total number of bits available for storing information, subtract the number of reserved bits from the total number in the grid. For Version 3 in Figure 1.22(a), that is 29 × 29 (or 841), minus 223 for the first group of regions, minus 26 for the timing patterns, minus 25 for the alignment pattern, which equals 567 bits.

The version information area

The version information area is only present on Version 7 and higher. If it is present, it occupies two 3 × 6 blocks adjacent to the upper-right and lower-left finder patterns. Figure 1.22(b) for the 45 × 45 grid of Version 7 shows the placement of these two regions, each of which occupies 18 bits. This version also has six alignment patterns, two of which intersect the timing patterns. Each intersecting alignment pattern takes away only 20 bits from the total instead of 25, because the middle row or column is already accounted for by the bits in the timing pattern. Note that the alternating one–zero pattern of the timing track is not disturbed by the intersecting alignment pattern.

Example 1.4 How many bits are available for storing information in the Version 7 QR code of Figure 1.22(a)?

number of bits available for information
= ⟨Subtract reserved areas from total⟩
$45 \times 45 - \mathit{fixed} - \mathit{timing} - \mathit{align} - \mathit{interalign} - \mathit{vinfo}$
= ⟨Substitute values⟩
$45 \times 45 - 223 - 2(45 - 2 \times 8) - 4 \times 25 - 2 \times 20 - 2 \times 18$
= ⟨Math⟩
1496 b

where *fixed* is the number of bits in the first four reserved regions, *timing* is the number of bits in the two timing patterns, *align* is the number of bits in the four alignment patterns that do not intersect a timing pattern, *interalign* is the number of bits in the two alignment patterns that do intersect a timing pattern, and *vinfo* is the number of bits in the two version information areas. ■

There are four kinds of information bits in a QR code.

The QR code information bits

- The mode indicator
- The character count indicator
- The redundant bits for error correction
- The data bits

The mode indicator

The mode indicator is a four-bit string that specifies what kind of characters the code represents. The numeric mode is for codes that store only numbers. The alphanumeric mode stores only sequences of uppercase

English characters, the 10 decimal digits, and a few punctuation characters. The byte mode stores sequences of ASCII characters. There are two other modes, one of which stores Kanji characters for Asian languages.

The character count indicator

The character count indicator contains anywhere from 8 to 16 bits, depending on the version and the mode. For example, in Version 10 using the numeric mode, the character count string is 12 bits long, but in Version 27 it is 14 bits long. Regardless of how many bits are in the character count indicator, it represents the number of characters stored in the code. The Version 3 QR code of Figure 1.21(a) uses the byte mode to store 43 ASCII characters of a URL address. Its character count indicator contains the 8 bits 0010 1011, because that is how to represent the integer 43 in binary.

Whenever information is stored or transmitted, it is subject to errors. For example, a wireless signal might be distorted as you move your mobile device around the room. Or, on a QR code there might be a smudge of dirt that obscures the pattern of the code. The scanner might read the dirty smudge over a white bit as a 1 when the bit should be a 0. A common technique to handle such errors is to add extra bits to the data that allow the receiver to detect if an error has occurred and, if so, to correct it. Section 9.4 shows how error detection and correction codes work. The QR code uses the same error correction technique that is used to read Blu-ray discs when scratches on the surface of the disc can cause errors.

The redundant bits of error correction

FIGURE 1.23 shows the four possible correction levels in a QR code. The lowest correction level, L, can recover all the text in a QR code even if 7% of the code is damaged. The next higher correction level, M, can recover all the text even if 15% of the code is damaged. The higest level, H, can correct 30%. The higher the correction level, the more redundant bits are required, and so the fewer data bits can be stored in a given QR version. Levels L and M are most common in practice. The bits for the mode and character count indicators and the redundant bits for error correction represent overhead for the encoded message and must be subtracted from the information bits

FIGURE 1.23
The four possible correction levels in a QR code.

Correction at Level	Will Correct the Data With
L	7% damaged.
M	15% damaged.
Q	25% damaged.
H	30% damaged.

to determine how many data bits are available to store the message of the QR code. For the byte mode that stores a message of ASCII characters, the overhead is about 20% for Level L and 40% for Level M.

Example 1.5 How many ASCII characters can be stored in a 29×29 Version 3 QR code with one alignment pattern that does not intersect a timing pattern using Level L error correction with 25% overhead?

First, compute the number of information bits:

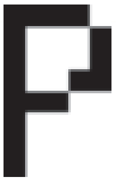
$$\begin{aligned}
& \text{number of bits available for information} \\
= \quad & \langle \text{Subtract reserved areas from total} \rangle \\
& 29 \times 29 - \textit{fixed} - \textit{timing} - \textit{align} \\
= \quad & \langle \text{Substitute values} \rangle \\
& 29 \times 29 - 223 - 2(29 - 2 \times 8) - 25 \\
= \quad & \langle \text{Math} \rangle \\
& 567 \text{ b}
\end{aligned}$$

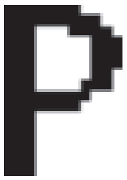
If the mode and character count indicators and the redundant bits for error correction have a 25% overhead, then $100\% - 25\%$, or 75% of the information bits are available for the data bits.

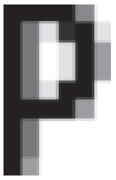
$$\begin{aligned}
& \text{maximum number of characters} \\
= \quad & \langle \text{Account for overhead} \rangle \\
& (\text{fraction for data}) \times (\text{number of characters}) \\
= \quad & \langle \text{Substitute values} \rangle \\
& (1.00 - 0.25) \times (567 \text{ b}/8(\text{b/char})) \\
= \quad & \langle \text{Math} \rangle \\
& 53.16 \\
= \quad & \langle \text{Round down} \rangle \\
& 53 \text{ characters}
\end{aligned}$$

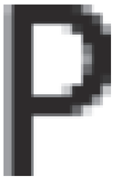
The URL in the QR code of Figure 1.21(a) is a Version 3 code with Level L error correction. It has 49 characters, which is just under the maximum limit of 53. ■

FIGURE 1.24
The letter *P* rendered in black and white and grayscale.


(a) Black and white, 5×8.


(b) Black and white, 11×16.


(c) Grayscale, 6×9.


(d) Grayscale, 11×17.

Images

Images in a computer system include images on computer screens, scanned images of paper documents, and photographic images captured by cameras. Depending on the device, the image may be black and white, grayscale, or color. Regardless of the device, all images in a computer system are stored in binary.

FIGURE 1.24 shows four renderings of the letter *P* on four different display devices. In part (a), the image of the letter occupies a 5- × 8-pixel region of the display, and in (b) it occupies an 11- × 16-pixel region. You can see from the two images that the more pixels that are available to store the image, the more accurate is the image. One way to increase the accuracy of the image is to increase the number of pixels per inch high enough that the human eye cannot perceive the individual pixels. Another way to increase the accuracy is to design the individual pixels with the ability to display shades of gray instead of just black and white. In part (c), the image of the letter has approximately the same number of pixels as in part (a), but some of the pixels have various shades of gray between pure black and pure white. Similarly, the image in part (d) has about the same number of pixels as the image in part (b) but is more accurate because of its grayscale, especially when viewed from a distance.

A black-and-white laser printer works on the same principle. It either does or does not deposit a dot of black toner on each pixel location on the paper. There is no gray toner, so product designers increase accuracy by increasing the number of dots per inch. A typical desktop laser printer can produce 600 or 1200 dots per inch. Commercial typesetting machines offer 2400 dots per inch or more. To print shades of gray, the printer deposits a pattern of black dots interspersed with white, which the human eye perceives as gray from a distance. Document scanners usually have the option to scan in black and white, grayscale, or color. When you view a scanned grayscale image on a screen, each screen pixel has the ability to display discrete shades of gray.

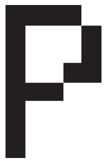
As with all information in a computer system, images are stored in binary. **FIGURE 1.25(a)** shows the binary storage for the black-and-white image in Figure 1.24(a). Each cell in the storage grid for the display contains a single bit that is 1 when the corresponding pixel is black and 0 when it is white. The total number of bits required to store the image is the total number of cells in the storage grid. The 5 × 8 grid requires 40 bits, which is 5 bytes, to store the image.

Bit depth

The *bit depth* of a stored image is the number of bits required to store a single pixel. In Figure 1.25(a), the bit depth is one because there is one bit per pixel. Figure 1.25(b) shows the binary storage for the grayscale image in Figure 1.24(c). The bit depth for this image is three because it takes three bits to store each pixel. The table in Figure 1.18 lists the eight possible values with a memory cell of three bits. Each pixel in the image of Figure 1.25(b) is one of eight possible shades of gray. For example, the pixel in the lower-left corner is black, with a binary value of 111, and the pixel in the lower-right corner is white, with a value of 000. The pixel in the bottom row, second from the left, has a binary value of 010, which produces a light gray color, and the pixel just above that is darker, with a binary value of 101. The total number of bits required to display a grayscale image is the number of pixels times the bit depth.

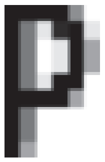
FIGURE 1.25
Binary storage for black-and-white images and for grayscale images.

1	1	1	1	0
1	0	0	0	1
1	0	0	0	1
1	0	0	1	1
1	1	1	0	0
1	0	0	0	0
1	0	0	0	0
1	0	0	0	0



(a) Storage for the image in Figure 1.24(a).

111	111	111	111	010	000
111	101	001	101	111	001
111	101	000	001	111	010
111	101	000	001	111	010
111	101	001	101	111	000
111	111	111	110	010	000
111	101	000	000	000	000
111	101	000	000	000	000
111	010	000	000	000	000



(b) Storage for the image in Figure 1.24(c).

Example 1.6 An eReader has a 1072- × 1448-pixel grayscale display, with each pixel able to display 16 shades of gray. What is the KiB size of the display memory for the device?

First, determine the bit depth of the display. Because each pixel can display 16 shades of gray, and $2^4 = 16$, the bit depth is 4. Then compute the size of the display memory as follows:

size of display memory
= ⟨Product⟩
(number of pixels) × (bit depth)
= ⟨Substitute values⟩
(1072 × 1448 pixels) × 4(b/pixel)
= ⟨Math⟩
6,209,024 b
= ⟨Convert to KiB⟩
(6,209,024 b) × (1 B/8 b) × (1 KiB/1024 B)
= ⟨Math⟩
758 KiB

In a color display, each pixel on the screen emits a color. The human eye captures the light rays from the field of pixels and focuses them on the retina. The retina contains two kinds of photoreceptor cells that are sensitive to light. It has about 6 million cone cells, which are sensitive to color, and 120 million rod cells, which are not sensitive to color and enable vision in low light conditions. The cells convert the light energy into electrical signals that are sent through the optic nerves to the brain. The brain combines all the signals from the photoreceptor cells to form an image in the mind. Strictly speaking, light rays do not have color. Color exists only in the human mind. When you view a photographic scene on a color display, the pixels emit points of light that are detected by the photoreceptor cells and processed by the brain so that the resulting image in the mind approximates the image that would be constructed by the brain were you to view the scene in nature.

Sunlight, which appears to be white in the human mind, is a mixture of a spectrum of colors. When the sun comes out on a rainy day, each droplet of water in the air is a tiny prism that separates the mixture of colors into the visible spectrum to make a rainbow. Each color in the spectrum is a pure color whose light rays have a single wavelength. FIGURE 1.26 shows the wavelengths of light in the visible spectrum, which ranges from about 400 nm (nanometers) at the violet end of the spectrum to 700 nm at the red end. Colors outside the spectrum are mixtures of pure colors. For example, if a light ray is a mixture of the pure colors red and blue, the brain will interpret the signals from the cones to produce the color purple in the mind.

The retina has three types of cone cells that are sensitive to short, medium, and long wavelengths of light in the visible spectrum. FIGURE 1.27 shows that each type of cone cell is sensitive to a range of wavelenths. The first type, known as an *S-cone* (for *short wavelength*), is sensitive to wavelengths of light between about 400 nm and 540 nm. It has peak sensitivity at about 430 nm, which corresponds to a blue-violet color. The peak sensitivity of the second, *M-cone* (for *medium wavelength*), is about 540 nm, corresponding roughly to green. The third, *L-cone* (for *long wavelength*), has the widest range of sensitivity and is more sensitive to red than the other cone types.

Figure 1.27 shows that if a source of light sends a pure 580-nm light ray to the human eye, both M-cones and L-cones will detect the ray and send their signals to the brain. The brain combines the signals from both types of cells to produce the color yellow in the mind. It is possible for two different light rays to produce the same color in the mind. For example, a light ray containing a mixture of pure red and pure green can also activate both M-cones and L-cones in such a way that their combined signals are identical to the signals sent by a pure 580-nm light ray. In that case, the mind will again perceive the color yellow.

Like grayscale displays, color displays are composed as a grid of pixels, with the difference that each pixel contains three subpixels: one that emits red light, one that emits green light, and one that emits blue light. FIGURE 1.28(a) shows the structure of a single-color pixel where R is the red subpixel, G is the green subpixel, and B is the blue subpixel. This layout of the subpixels, known as *RGB stripe*, is the most common, although some devices have a different arrangement of subpixels. A display is a two-dimensional grid of square pixels. Figure 1.28(b) is a 16 × 8 portion of a color display that shows the individual subpixels in each pixel. From a distance, the eye cannot

FIGURE 1.26
The colors of the visible spectrum.

Color	Wavelength
Violet	400–450 nm
Blue	450–495 nm
Green	495–570 nm
Yellow	570–590 nm
Orange	590–620 nm
Red	620–700 nm

FIGURE 1.27
Cone cell sensitivity as a function of wavelength.

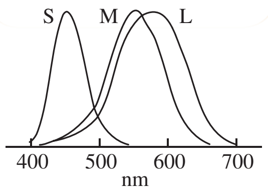
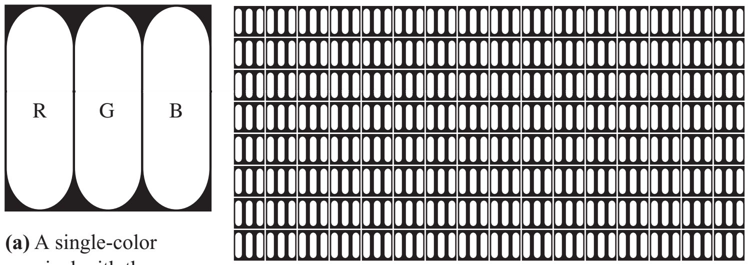


FIGURE 1.28
Color pixels.



(a) A single-color pixel with three subpixels.

(b) A 16- × 8-pixel portion of a color display.

distinguish between individual subpixels. The cone cells receive the light from a pixel as if it were a single source of light, with its color determined by the mixture of its red, green, and blue components.

When we look at our environment in nature, our eyes do not receive mixtures of only red, green, and blue. They receive mixtures of all the wavelengths from the visible spectrum. However, when we look at a scene on a computer display, our eyes receive mixtures of only the red, green, and blue from the subpixels. A scene on a display can appear realistic because the combination of red, green, and blue for each pixel produces the sensation of color in our minds that approximates the sensation produced by a different mixture of wavelengths in the real world. However, it is impossible with only three subpixels to produce all the colors the human eye can see in nature. The only colors a human can see on a display are those that are produced in our mind by mixtures of the colors from the red, green, and blue subpixels.

As with a grayscale image, the number of bits required to store a color image is equal to the total number of pixels in the display times the bit depth of each pixel. In Figure 1.25(b), the bit depth of the grayscale image is 3 because each bit can display eight shades of gray. Figure 1.28(a), however, shows that each color pixel consists of three subpixels. If each subpixel can display eight shades of its color, the total bit depth is 3 bits for each pixel times 3 pixels, which is 9. The total bit depth for a color pixel is sometimes referred to as the *color depth*. Color displays can typically produce 256 levels of brightness for each subpixel. Because 2^8 is 256, such displays require 8 bits each for the R, G, and B subpixels, for a total bit depth of 24. FIGURE 1.29 shows some of the colors possible for such a pixel. The table shows the brightness levels in decimal, but as with all information in a computer system, they are physically stored as bits. A brightness of 255 is stored with 8 bits in binary as 1111 1111, a brightness of 192 is stored as 1100 0000, and a brightness of 128 is stored as 1000 0000. Chapter 3 describes these equivalences between decimal and binary.

FIGURE 1.29
Some colors produced by a color pixel with a bit depth of 24.

Color	Red	Green	Blue
White	255	255	255
Silver	192	192	192
Gray	128	128	128
Black	0	0	0
Red	255	0	0
Maroon	128	0	0
Yellow	255	255	0
Olive	128	128	0
Lime	0	255	0
Green	0	128	0
Aqua	0	255	255
Teal	0	128	128
Blue	0	0	255
Navy	0	0	128
Fuchsia	255	0	255
Purple	128	0	128

Example 1.7 A GPS system in an automobile has a 4.5- × 2.5-inch screen with 120 color pixels per inch. Each subpixel color can display 64 levels of brightness. What is the KiB size of the display memory?

First, determine the total number of pixels in the display.

number of pixels
= ⟨Product⟩
(number in width) × (number in height)
= ⟨Substitute values⟩
(4.5 in. × 120 pixels/in.) × (2.5 in. × 120 pixels/in.)
= ⟨Math⟩
162,000 pixels

Because each subpixel can display 64 levels of brightness, and $2^6 = 64$, the number of bits for each subpixel is 6. Because there are 3 subpixels per pixel, the bit depth is 3×6, which is 18. Compute the size of the display memory as follows:

size of display memory
= ⟨Product⟩
(number of pixels) × (bit pixels)
= ⟨Substitute values⟩
(162,000 pixels) × 18(b/pixel)
= ⟨Math⟩
2,916,000 b
= ⟨Convert to KiB⟩
(2,916,000 b) × (1 B/8 b) × (1 KiB/1024 B)
= ⟨Math⟩
356 KiB ■

The fact that a color display cannot produce all the colors of nature but only an approximation of them is typical for all aspects of computer systems. Computers are useful because they perform tasks from the real world. In so doing, they *model* aspects of the real world. But computer models of the real world are always approximations.

For example, in a color display with a bit depth of 24, each pixel can produce 2^{24}, or 16,777,216, different colors. The human eye can distinguish about 10,000,000 colors, so it would seem that a color display can produce all the colors perceptible by the human eye. Yet it cannot, for two reasons. First, the human eye cannot distinguish between the color of one pixel and that of another with only a single level of brightness different in one subpixel. Consequently, the number of distinguishable colors from the set of 16,777,216 is less than the 10,000,000 distinguishable by the human eye. Second, the only way to produce all the colors perceptible by the human eye would be to have more than three subpixels per pixel and to have their wavelengths span the entire visible spectrum.

As another example, one mathematical property of integers is that there is no largest value. Consequently, there are an infinite number of integer values. However, all computers are finite. If there are n bits in a memory cell that stores an integer, then only 2^n values can be stored, not an infinite number of values. For a given storage cell, there is a largest integer value. So, the properties of integers stored in computers do not quite match the properties of mathematical integers.

The Intel Core i7 System

This text is an introduction to computer systems at all seven levels of abstraction. In practice, computer systems are complex and contain a massive amount of detail at each level—far too much detail to describe in an introductory text. Consequently, much of the detail at each level is hidden. The text presents a simplified model of each level that illustrates the fundamental principles that operate at that level. The principles govern all physical computer systems, but the model itself is only an approximation of systems in practice. This section is the first of a series of sidebars created to describe some of the omitted details and to show how the principles apply to systems in practice.

In the early history of computers, different manufacturers designed and sold many different computer systems. Over time, however, the number of different computer systems dwindled until the present, when two computer systems dominate the commercial market—namely, the Intel/AMD systems and the ARM systems. Intel and Advanced Micro Devices (AMD) are the manufacturers of the x86 series of computer systems common in desktops and laptops. For mobile devices like smartphones and tablets, however, the Advanced RISC Machines (ARM) systems dominate the market. The *R* in the acronym *ARM* is itself an acronym and stands for *Reduced Instruction Set Computer (RISC)*. In contrast, the x86 series of computers use the Complex Instruction Set Computer (CISC) design. Chapter 12 describes the difference between CISC and RISC designs.

The Intel Core i7 system is a set of integrated circuits, also called *computer chips*, that form the basis of many Microsft Windows and Apple OS X desktops and laptops. FIGURE 1.30 shows some of the details of the Core i7 system that are not present in the corresponding

FIGURE 1.30
The Intel Core i7 system.

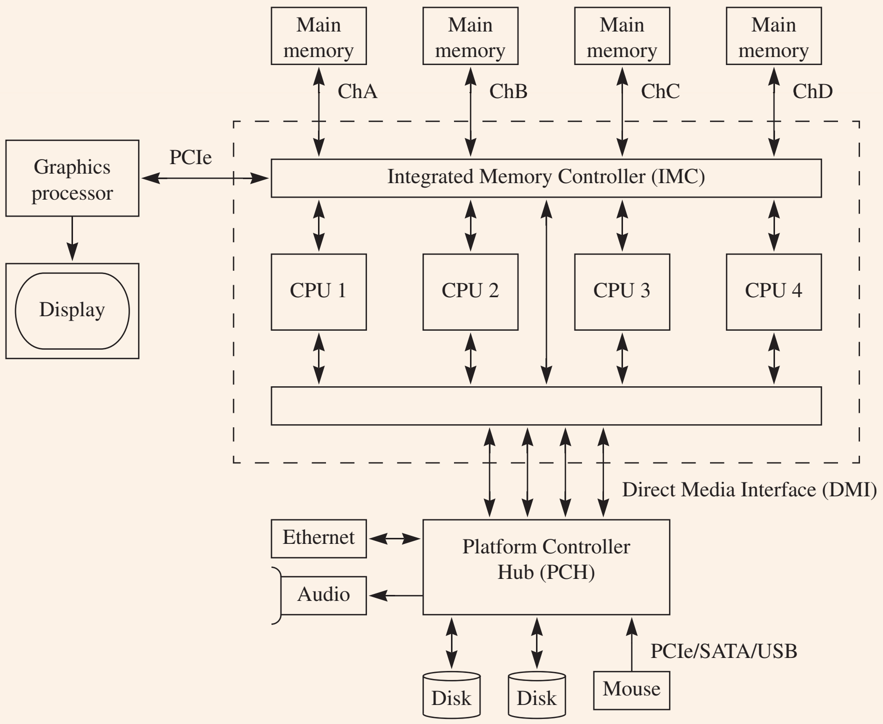


Figure 1.9. The dashed box represents a single physical package about 2 inches square that mounts on a circuit board. The integrated circuit inside the package has connections to a grid of pins on the bottom of the package that make electrical contact with the surface of the circuit board. The Platform Controller Hub (PCH) is another part mounted on the circuit board.

The most significant difference between the simplified model of Figure 1.9 and current systems is the existence of more than one CPU. A single chip typically contains several central processing units called *cores* that execute simultaneously. Having more than one CPU executing at the same time speeds up computations, much in the way that having several people working simultaneously on a single task shortens the time to complete the task. The Core i7 comprises a series of models, with the number of cores ranging from four, as in Figure 1.30, to eight, depending on the model.

Figure 1.9, which represents one primary system bus shared by all peripheral devices and main memory, is characteristic of early personal computer systems. The arrows labeled *Direct Media Interface (DMI)* in Figure 1.30 correspond to the system bus in Figure 1.9. All the peripheral devices, such as the disk drives and the Ethernet connection to the Internet, share the bus, which is connected to the CPUs on the neighboring package. The PCH controls the communication on the DMI bus, switching between the devices and scheduling the communication over the bus. In contrast to the system depicted in Figure 1.9, the main memory modules do not share the DMI system bus but are instead connected to the CPUs with separate paths called *channels*. The four memory channels are labeled *ChA*, *ChB*, *ChC*, and *ChD*. The integrated memory controller (IMC) inside the Core i7 package can connect any memory module to any CPU. Furthermore, each CPU cannot distinguish between individual main memory modules. The IMC makes all the memory modules appear like a single main memory to each CPU.

The connection between different subsystems on the circuit board is with one of several industry-standard buses: Peripheral Component Interconnect Express (PCIe), Serial Advanced Technology Attachment (SATA), or Universal Serial Bus (USB). PCIe usually connects two subsystems that are both permanently mounted on the circuit board. SATA is a common bus for connecting external disk drives to the circuit board. The USB bus is common for smaller peripherals like track pads and thumb drives. Two other common buses not shown in Figure 1.30 are Thunderbolt and High-Definition Multimedia Interface (HDMI). Thunderbolt is a combination of PCIe and another bus called *DisplayPort*, which was originally developed for video data. It is a high-speed bus common for transferring large amounts of data, as required, for example, in video-editing systems. HDMI is an interface for transferring digital video and audio signals between the components of a system. It was originally designed to be the primary interconnection link in home entertainment systems but is now common as an output port for computer systems as well.

1.5 Database Systems

Database systems are one of the most common applications at Level App7. A *database* is a collection of files that contain interrelated information, and a *database system* (also called a *database management system*, or *DBMS*) is a program that lets the user add, delete, and modify records in the database. A database system also permits queries of the database. A *query* is a request for information, usually from different parts of the database.

An example of a database is the information an online retailer maintains about the inventory, prices, product descriptions, and orders for its merchandise. A query might be a request for a listing showing the number of orders for all the items with priority shipping to be sent to a particular country at the end of the day. To produce the listing, the database system combines the information from different parts of the database, in this case from an order file and from a customer address file.

Relations

Relational database systems

Relational database systems store information in files that appear to have a table structure. Each table has a fixed number of columns and a variable number of rows. FIGURE 1.31 is an example of the information in a relational database. Each table has a name. The table named `Sor` contains information about the members of a sorority, and the one named `Frat` contains information about the members of a fraternity. The user at Level App7 fixed the number of vertical columns in each table before entering the information in the body of the tables. The number of horizontal rows is variable so that individuals can be added to or deleted from the tables.

`Sor`

S.Name	**S.Class**	**S.Major**	**S.State**
Beth	Soph	Hist	TX
Nancy	Jr	Math	NY
Robin	Sr	Hist	CA
Allison	Soph	Math	AZ
Lulwa	Sr	CompSci	CA

`Frat`

F.Name	**F.Major**	**F.State**
Emile	PolySci	CA
Sam	CompSci	WA
Ron	Math	OR
Mehdi	Math	CA
David	English	AZ
Jeff	Hist	TX
Craig	English	CA
Gary	CompSci	CA

FIGURE 1.31
An example of a relational database. This database contains two relations—`Sor` and `Frat`.

Relations, attributes, tuples, and domains

In relational database terminology, a table is called a *relation*. A column is an *attribute*, and a row is a *tuple* (rhymes with *couple*). In Figure 1.31, `Sor` and `Frat` are relations, (Nancy, Jr, Math, NY) is a 4-tuple of `Sor` because it has four elements, and `F.Major` is an attribute of `Frat`. The *domain* of an attribute is the set of all possible values of the attribute. The domain of `S.Major` and `F.Major` is the set {Hist, Math, CompSci, PolySci, English}.

Queries

Examples of queries from this database are requests for Ron's home state and for the names of all the sophomores in the sorority. Another query is a request for a list of those sorority and fraternity members who have the same major, and what that common major is.

In this small example, you can manually search through the database to determine the result of each of these queries. Ron's home state is OR, and Beth and Allison are the sophomores in the sorority. The third query is a little more difficult to tabulate. Beth and Jeff are both history majors. Nancy and Ron are both math majors, as are Nancy and Mehdi. Robin and Jeff are both history majors, and so on.

It is interesting that the result of each of these queries can be written in table form as in FIGURE 1.32. The result of the first query is a table with one column and one row, while the result of the second is a table with one column and two rows. The result of the third is a table with three columns and eight rows. So the result of a query of a relational database, which is a collection of relations, is itself a relation!

FIGURE 1.32 The result of three queries from the database of Figure 1.31. Each result is a relation.

`Result1`

F.State
OR

`Result2`

S.Name
Beth
Allison

`Result3`

S.Name	**F.Name**	**Major**
Beth	Jeff	Hist
Nancy	Ron	Math
Nancy	Mehdi	Math
Robin	Jeff	Hist
Allison	Ron	Math
Allison	Mehdi	Math
Lulwa	Sam	CompSci
Lulwa	Gary	CompSci

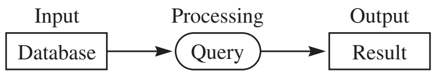


FIGURE 1.33 The relationship between the database, a query, and the result.

Query as a relation result

The fact that the result of a query is itself a relation is a powerful idea in relational database systems. The user at Level App7 views the database as a collection of relations. Her query is a request for another relation that the system derives from the existing relations in the database.

Remember that each level has a language. The language of a Level App7 relational DBMS is a set of commands that combines or modifies existing relations and produces new relations. The user at Level App7 issues the commands to produce the desired result. FIGURE 1.33 shows the relationship between the database, a query, and the result. The database is the input. The query is a set of commands in the Level App7 language. As it does in every level in the computer system, the relationship takes this form: input, processing, output.

This chapter cannot describe every language of every relational database system on the market. Most of these lanugages are variations on a standard known as *Structured Query Language (SQL)*. Instead, it describes a simplified language typical of such systems. Most relational DBMS languages have many powerful commands. But three commands are fundamental—`select`, `project`, and `join`.

The `select` and `project` statements are similar because they both operate on a single relation to produce a modified relation. The `select` statement takes a set of rows from a given table that satisfies the condition specified in the statement. The `project` statement takes a set of columns from a given table according to the attributes specified in the statement. FIGURE 1.34 illustrates the effect of the statements

```
select Frat where F.Major = English giving Temp1
```

and

```
project Sor over S.Name giving Temp2
```

The `project` statement can specify more than one column, in which case the attributes are enclosed in parentheses and separated by commas. For example,

```
project Sor over (S.Class, S.State) giving Temp3
```

selects two attributes from the `Sor` relation.

FIGURE 1.34
The `select` and `project` operators.

Temp1

F.Name	F.Major	F.State
David	English	AZ
Craig	English	CA

(a) `Select Frat where F.Major = English giving Temp1`

Temp2

S.Name
Beth
Nancy
Robin
Allison
Lulwa

(b) `Project Sor over S.Name giving Temp2`

Temp3

S.Class	S.State
Soph	TX
Jr	NY
Sr	CA
Soph	AZ

(c) `Project Sor over (S.Class, S.State) giving Temp3`

Note in Figure 1.34(c) that the pair (Sr, CA) is common from both 4-tuples (Robin, Sr, Hist, CA) and (Lulwa, Sr, CompSci, CA) in relation `Sor` (Figure 1.31). But the pair is not repeated in relation `Temp3`. A basic property of relations is that no row in any table may be duplicated. The `project` operator checks for duplicated rows and does not permit them. Mathematically, a relation is a set of tuples, and elements of a set cannot be duplicated.

`join` differs from `select` and `project` because its input is two tables, not one. A column from the first table and a column from the second table are specified as the `join` column. The `join` column from each table must have a common domain. The result of a `join` of two tables is one wide table whose columns are duplicates of the original columns, except that the `join` column appears only once. The rows of the resulting table are copies of those rows of the two original tables that have equal elements in the `join` column.

For example, in Figure 1.31 the columns `S.Major` and `F.Major` have a common domain. The statement

```
join Sor and Frat over Major giving Temp4
```

specifies that `Major` is the `join` column and that the relations `Sor` and `Frat` are to be joined over it. FIGURE 1.35 shows that the only rows included in the `join` of the two tables are the ones with equal majors. The 4-tuple (Robin, Sr, Hist, CA) from `Sor` and the 3-tuple (Jeff, Hist, TX) from `Frat` are joined in `Temp4` because their majors, Hist, are equal.

Temp4

S.Name	S.Class	S.State	Major	F.Name	F.State
Beth	Soph	TX	Hist	Jeff	TX
Nancy	Jr	NY	Math	Ron	OR
Nancy	Jr	NY	Math	Mehdi	CA
Robin	Sr	CA	Hist	Jeff	TX
Allison	Soph	AZ	Math	Ron	OR
Allison	Soph	AZ	Math	Mehdi	CA
Lulwa	Sr	CA	CompSci	Sam	WA
Lulwa	Sr	CA	CompSci	Gary	CA

FIGURE 1.35
The `join` operator. The relation is from the statement `join Sor and Frat over Major giving Temp4`.

Structure of the Language

The statements in this Level App7 language have the following form:

`select` *relation* `where` *condition* `giving` *relation*
`project` *relation* `over` *attributes* `giving` *relation*
`join` *relation* `and` *relation* `over` *attribute* `giving` *relation*

The reserved words of the language are

Reserved words

```
select    project
join      and
where     over
giving
```

Each reserved word has a special meaning in the language, as the previous examples demonstrate. Words to identify objects in the language, such as `Sor` and `Temp2` to identify relations and `F.State` to identify an attribute, are not reserved. They are created arbitrarily by the user at Level App7 and are called *identifiers*. The existence of reserved words and user-defined identifiers is common in languages at all the levels of a typical computer system.

Do you see how to use the `select`, `project`, and `join` statements to generate the results of the query in Figure 1.32? The statements for the first query, which asks for Ron's home state, are

```
select Frat where F.Name = Ron giving Temp5
project Temp5 over F.State giving Result1
```

The statements for the second query, which asks for the names of all the sophomores in the sorority, are

```
select Sor where S.Class = Soph giving Temp6
project Temp6 over S.Name giving Result2
```

The statements for the third query, which asks for a list of those sorority and fraternity members who have the same major and what that common major is, are

```
join Sor and Frat over Major giving Temp4
project Temp4 over (S.Name, F.Name, Major) giving
Result3
```

Chapter Summary

The fundamental question of computer science is: What can be automated? Computers automate the processing of information. The theme of this text is levels of abstraction in computer systems. Abstraction includes suppression of detail to show the essence of the matter, an outline structure, division of responsibility through a chain of command, and subdivision of a system into smaller systems. The seven levels of abstraction in a typical computer system are

Level 7 (App7):	Application
Level 6 (HOL6):	High-order language
Level 5 (Asmb5):	Assembly
Level 4 (OS4):	Operating system
Level 3 (ISA3):	Instruction set architecture
Level 2 (Mc2):	Microcode
Level 1 (LG1):	Logic gate

Each level has its own language, which serves to hide the details of the lower levels.

A computer system consists of hardware and software. Three components of hardware are the central processing unit, main memory, and disk storage. Of the three components, the disk has the highest storage capacity but is the slowest, and the CPU has the smallest capacity but is the fastest. Main memory is between the two in both capacity and speed.

Programs that control the hardware are called *software*. An algorithm is a set of instructions that, when carried out in the proper sequence, solves a problem in a finite amount of time. A program is an algorithm written for execution on a computer. A program inputs information, processes it, and outputs the results. Software analysis determines the output, given the input and the program. Software design determines the program, given the input and the desired output. The operating system is a large program that provides the human interface of the computer to the user. It manages files, memory, and processors.

The smallest unit of digital information is the binary digit, or bit. Each bit in a computer system occupies space. A computer system stores all information, such as numbers, text, and images, as collections of bits. A collection of eight bits is a byte. The number of values stored by a sequence of n bits is 2^n. It takes one byte, which is eight bits, to store one ASCII character.

All computations in a computer system occupy time as well as space. The two components of time in a computer system are the computation time, which is the time it takes the CPU to execute one of the instructions in its instruction set, and the transmission time, which is the time it takes to move information from one component of the system to another. The following system performance equation computes the total execution time of the program task as the product of three terms:

$$\frac{\text{time}}{\text{program}} = \frac{\text{instructions}}{\text{program}} \times \frac{\text{cycles}}{\text{instruction}} \times \frac{\text{time}}{\text{cycle}}$$

The following bandwidth equation computes the total information transferred as the product of two terms:

$$\text{information} = \frac{\text{information}}{\text{time}} \times \text{time}$$

The first term in the product is the bandwidth of the channel, and the second term is the transmission time.

A QR code stores a bit of information in a square on the QR grid. A dark square is the binary value 1, and a light square is the binary value 0. The more squares in the grid, the more information the QR code contains. An image in a computer system is a grid of pixels. A black-and-white image requires one bit per pixel. The bit depth of a grayscale image is the number of bits per pixel, which determines how many shades of gray each pixel can display. Color images require three subpixels—one red, one green, and one blue—for each color pixel.

Database systems are one of the most common applications at Level App7. Relational database systems store information in files that appear to have a table structure; this table is called a *relation*. The result of a query in a relational database system is itself a relation. The three fundamental operations in a relational database system are `select`, `project`, and `join`. A query is a combination of these three operations.

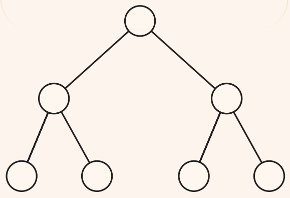
Exercises

At the end of each chapter in this text is a set of exercises and problems. Work the exercises on paper by hand. Answers to the starred exercises are in the back of the text. (For some multipart exercises, answers are supplied only for selected parts.) The problems are programs to be entered into the computer. This chapter contains only exercises.

Section 1.1

1. **(a)** Draw a hierarchy diagram that corresponds to the United States Constitution. **(b)** Based on Figure 1.5, draw a nesting diagram that corresponds to the organization of the hypothetical publishing company.
2. Genghis Khan organized his men into groups of 10 soldiers under a "leader of 10." Ten "leaders of 10" were under a "leader of 100." Ten "leaders of 100" were under a "leader of 1000." ***(a)** If Khan had an army of 10,000 soldiers at the lowest level, how many men in total were under him in his organization? **(b)** If Khan had an army of 5763 soldiers at the lowest level, how many men in total were under him in his organization? Assume that the groups of 10 should contain 10 if possible, but that one group at each level may need to contain fewer.
3. In the Bible, Exodus Chapter 18 describes how Moses was overwhelmed as the single judge of Israel because of the large number of trivial cases that were brought before him. His father-in-law, Jethro, recommended a hierarchical system of appellate courts where the lowest-level judge had responsibility for 10 citizens. Five judges of 10 sent the difficult cases that they could not resolve to a judge of 50 citizens. Two judges of 50 were under a judge of 100, and 10 judges of 100 were under a judge of 1000. The judges of 1000 citizens reported to Moses, who had to decide only the most difficult cases. ***(a)** If the population were exactly 2000 citizens (excluding judges), draw the three top levels of the hierarchy diagram. **(b)** In part (a), what would be the total population, including Moses, all the judges, and citizens? **(c)** If the population were exactly 10,000 citizens (excluding judges), what would be the total population, including Moses, all the judges, and citizens?
4. A full binary tree is a tree whose leaves are all at the same level, and every node that is not a leaf has exactly two nodes under it. FIGURE 1.36 is a full binary tree with three levels. ***(a)** Draw the full binary tree with four levels. ***(b)** How many nodes total are in a full binary tree with five levels? **(c)** With six levels? **(d)** With *n* levels in general?

FIGURE 1.36
Exercise 4: The full binary tree with three levels.



Section 1.2

5. True or false? **(a)** Main memory is volatile. **(b)** Main memory is accessed sequentially. **(c)** Disk memory is volatile. **(d)** Disk memory is accessed sequentially. **(e)** Main memory has greater storage capacity than disk memory. **(f)** Main memory has faster access time than disk memory.

Section 1.3

6. **(a)** What is an algorithm? **(b)** What is a program?

7. Answer the following questions about your operating system. **(a)** What is the name of your operating system? **(b)** Are certain characters disallowed or problematic in the names of the files? **(c)** Does your operating system distinguish between uppercase and lowercase characters in a file name?

8. Determine how to perform each of the following procedures with your operating system. **(a)** Set up a new user account. **(b)** Display the names of the files and subdirectories at the root directory. **(c)** Delete a file from the disk. **(d)** Change the name of a file. **(e)** Duplicate a file. **(f)** Display the size of a file. **(g)** Display the most recent modification time of a file.

Section 1.4

***9.** If an app requires the execution of 20 million instructions to complete a task and your CPU is rated at 2.1 GHz, what is the execution time of the task? Assume that each ISA3 instruction executes an average of 4.5 Mc2 instructions.

10. If an app requires the execution of 30 million instructions to complete a task and your CPU is rated at 2.8 GHz, what is the execution time of the task? Assume that each ISA3 instruction executes an average of 7.3 Mc2 instructions.

11. How long would it take to transfer a 600-MB database from disk to memory over a DMA channel with a bandwidth of 2.5 GB/s?

***12.** A typist is entering text on a keyboard at the rate of 40 words per minute. If each word is 5 characters long on average, what bandwidth in bits per second between the keyboard and main memory is required to transfer the information? A space is also a character. Assume that each word is followed by one space on average.

13. A typist is entering text on a keyboard at the rate of 30 words per minute. If each word is 6 characters long on average, what bandwidth in bits per second between the keyboard and main memory is required to transmit the information? A space is also a character. Assume that each word is followed by one space on average.

14. How many total bits are stored in the grid of a Version 4 QR code?

***15.** **(a)** How many bits are available for storing information in a 49 × 49 Version 8 QR code? This version has four alignment patterns that do not intersect a timing pattern and two that do intersect a timing pattern. It also has two 18-bit version information areas. **(b)** If the overhead for the mode, character count, and Level M error correction is 37%, how many characters can be stored in the code?

16. **(a)** How many bits are available for storing information in a 57 × 57 Version 10 QR code? This version has four alignment patterns that do not intersect a timing pattern and two that do intersect a timing pattern. It also has two 18-bit version information areas. **(b)** If the overhead for the mode, character count, and Level L error correction is 22%, how many characters can be stored in the code?

17. **(a)** A desktop laser printer has a 300 dots-per-inch resolution. If each dot is stored in one bit of memory, how many MiB of memory are required to store the complete image of one 8.5- × 11-inch page of paper? **(b)** How many MiB of memory are required for a 1200 dots-per-inch printer?

18. An eReader has a 956- × 1290-pixel grayscale display with each pixel able to display 32 shades of gray. What is the KiB size of the display memory for the device?

19. A mobile phone has a 3.48- × 1.96-inch screen size with a resolution of 326 pixels per inch. **(a)** How many pixels does it have? **(b)** With 256 levels of brightness for each color of subpixel, what is the MiB size of the display memory?

20. A tablet has a 7.5- × 5.8-inch screen size with a resolution of 326 pixels per inch. **(a)** How many pixels does it have? **(b)** With 256 levels of brightness for each color subpixel, what is the MiB size of the display memory?

Section 1.5

***21.** Write the relations `Temp5` and `Temp6` from the discussion in Section 1.5 of the chapter.

22. Write the statements for the following queries of the database in Figure 1.31. ***(a)** Find Beth's home state. **(b)** List the fraternity members who are English majors. **(c)** List the sorority and fraternity members who have the same home state, and indicate what that home state is.

23. **(a)** Write the statements to produce `Result2` in Figure 1.32, but with the `project` command before the `select`. **(b)** Write the statements to produce `Result3` in Figure 1.22, but with `join` as the last statement.