CHAPTER 1

Introduction

Computer science deals with the representation, implementation, manipulation and communication of information in digital format. The information can be viewed in different levels of abstractions, which are layered going from simple sequences of bits to complex cognitive processes. At each level of abstraction Computer Science has developed models to organize information effectively, called data structures, to process the information efficiently, called algorithms, and to communicate among the various levels, called protocols.

Before we can explore the details of the discipline of computer science, we need to know something about computer system structure. We begin by discussing the basic functions of a computer system. We also describe the basic computer architecture that makes it possible to store, retrieve, and manipulate digital information.

Because information is so important, computer scientists have developed a large body of concepts and techniques for manipulating data. These concepts and techniques form the focus of this book. This chapter briefly introduces the principles of Computer Science.

CHAPTER OBJECTIVES

- To provide a gentle introduction to the world of computing.
- To provide coverage of basic computer system structure.
- To provide coverage of basic computer science.

1.1 The Age of Digital Computing

In the last five decades of the twentieth century, we have witnessed a technology revolution which is primarily fueled by the invention of the **digital computer**. These marvelous devices have literally changed the way we live. What are these devices and why do they have such a big impact on our society?

A digital computer is a general-purpose computing device. It can manipulate numbers (integers and real), symbols (alphanumeric as well as special symbols like ".", "$", etc.), audio signals, and video signals. These abilities make these computing devices not only suitable for the traditional role of "computers" but also ideal devices for being embedded in all type of general-purpose appliances, including:

- Home appliances—refrigerators, washing machines, coffee makers, alarm systems, etc.
- Office appliances—FAX machines, copy machines, etc.
- Communication appliances—phones, cell phones, PDAs, GPSs.
- Entertainment appliances—digital radios, CD players, DVD players, Digital TV, etc.
- Vehicles—Antilock Breaks.
- Airplanes—Automatic Pilot.
- Ships—Automatic Pilot.

Indeed, most of the major infrastructures that we are totally relying on these days are controlled by computing devices, including

- Telephone network
- Power grid
- Air traffic control system.

This technology revolution did not start overnight. In the early days, very few people worked directly with computing devices, although without realizing it they interacted with computers indirectly—through printed reports such as credit card statements, or through agents such as bank tellers and airline reservation agents. Then, personal computers came along allowing ordinary people to start interacting with and using computers. Interfaces, such as phone interfaces to computers (interactive voice response systems) also allowed users to deal directly with a remote computer—a caller could dial a number, and press phone keys to enter information or to select alternative options, to find flight arrival/departure times, for example, or to register for courses in a university.

The Internet revolution of the late 1990s sharply increased direct user access to **remote computers**—computers that belong to a different organization and are housed in a different location than the one where the local computer resides. Organizations converted many of their phone interfaces to stored data into Web interfaces, and made a variety of services and information available online. For instance, when you access an online bookstore and browse a book or music collection, you are accessing data stored in a remote computer. When you enter an order online, your order is stored in that remote computer. When you access a bank Web site and retrieve your bank balance and transaction information, the information is retrieved from the bank's computer system. When you access a Web site, information about you may be retrieved from a

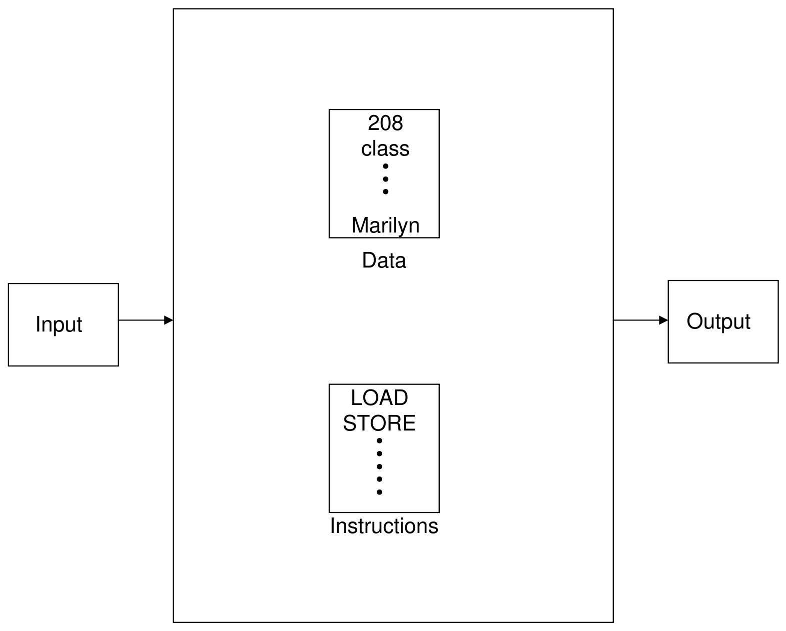

Figure 1.1 A simple view of a computing device.

computer to select which advertisements you should see. Furthermore, data about your Web accesses may be stored in a remote computer.

Thus, although user interfaces hide details of access to remote computers, and most people are not even aware they are dealing with remote computers, accessing remote computer forms an essential part of almost everyone's life today.

Moreover, many of the computing devices in the world are embedded in other devices and are not directly accessible by the ordinary user. As an example, consider the computing device that controls the breaking system in your car to ensure that your car does not skid; this devise is not directly accessible by you.

1.2 The Incredible Digital Computer

A computing device, in its simplest form, consists of two major parts (see Figure 1.1):

- **Data**—a set of information such as numbers, text, audio, and video.
- **Instructions**—a set of commands to manipulate the data, input new data (from a keyboard or other input devices), and output some data (to a printer or other output devices).

As an example, consider a simple computer whose only function is to serve as a personal calendar. The data stored consists of information about the 365 days in a year and the set of appointments one has made. The input device is a small keyboard that allows to input text about appointments (e.g, Lunch at 12:30 PM on July 9, 2006). The output device is the computer screen. Some of the instructions are commands to store new appointments, to figure out the current date, to update information (e.g., change the Lunch appointment on July 9, 2006 from at 12:30 PM to 1:00 PM), etc.

As another example, consider a typical laptop. The data stored on a laptop is usually much larger and richer in nature than the one of the "personal calendar" device. It consists of Word documents, JPEG images, MP3 audio, DVD movies, e-mail, etc. The input device is the keyboard that allows to input data (usually alphanumeric). There maybe additional input devices attached to your laptop (e.g., a mouse, a microphone). There are usually multiple output devices: the computer screen, the speakers, printer).

There are many other computing devices that have the same general structure, but with different capabilities. Here is a partial list.

- Hand-held calculator.
- CD player
- DVD player
- Real-time stock ticker
- Cell phone
- Pagers

The amazing aspect is that all these seemingly different devices share the same basic structure. It is our aim in this text to explain to you what this basic structure is all about and take you on an exciting journey into the digital world.

1.2.1 The Binary World

What is unique about these computing devices is that they are based on a very simple scheme—both the hardware and software are digital-based. That is, the environment they are operating in assumes that the world consists of only two states, commonly referred to as **binary** and represented by the symbols "0" and "1".

As we shall see, all information in a computer can be represented with these two symbols. These two symbols are called **bits**, standing for binary digits. A sequence of 8 bits, is called a **byte**. An example of a byte is the sequence:

$$01001101$$

A sequence of 4 bytes (or 32 bits) is called a **word**. An example of a word is the sequence:

$$01001101100011110000111110101010$$

You should not worry now about the meaning of these binary sequences. This will be explained later on in the text.

When discussing the size of (data) information, the usual counting matrix is a byte. A kilobyte, or KB, is 1,024 bytes; a megabyte, or MB, is $1{,}024^2$ bytes;

a gigabyte, or GB, is 1,024^3 bytes; and a terabyte, or TB, is 1,024^4. Computer manufacturers often round off these numbers and say that a megabyte is 1 million bytes and a gigabyte is 1 billion bytes.

1.2.2 Computer-System Structure

You now have a good basic understanding of what a computer, in general, looks like. It is time to dig deeper and get a closer more detailed look at the internals. A computing device consists of four basic components.

- **Main Memory**—The place where the data and instructions to manipulate the data are stored. Main memory is also referred to as **random-access memory** (or **RAM**)
- **Central Processing Unit (CPU)**—The control unit that executes the instructions.
- **Input/Output (I/O) controllers**—The control units for the various peripheral devices (where data is stored and displayed/printed).
- **Bus**—The communication structure for connecting the main memory, CPU, and the I/O controllers.

Figure 1.2 depicts the general structure of a modern general-purpose computer system, which consists of one or more CPUs and a number of device controllers connected through a common bus that provides access to shared memory. This memory is electrically a matrix of storage cells ("machine word", usually 4 consecutive bytes) that are addressed using the physical coordinates. The address is conveyed from the CPU via the address bus to the RAM. A storage cell consists of an electronic circuit that is in either one of two states (low voltage for "0" and high voltage for "1"). The circuit will remain in its current state until a new state is stored.

The CPU contains integrated circuits that can execute the "machine instructions" (see Section 1.2.3) on the electrical level. The result is temporarily hold in registers and eventually stored back in main memory.

The I/O controllers are electronic boards that contain the logic to control the peripheral devices and pass data forth and back between the bus and the peripheral devices. Sometimes one controller can manage more than one device. This is the case in Figure 1.1 with the USB (Universal Serial Bus) controller.

Each device controller is in charge of a specific type of device (for example, disk drives, audio devices, and video displays). Special-purpose computing devices—those that have specific limited function (for example, the device in a washing machine), will have, in general, a much simpler structure.

The system bus contains wires for data and address. Using the address of a device is necessary because all controllers are connected in parallel to the bus. The address is used to select a specific device. Only the corresponding I/O controller will respond to a certain device address. This ensures that at any time only one device is active sending or receiving data via the system bus. To ensure orderly access to the shared memory, a memory controller is provided whose function is to synchronize access to the memory.

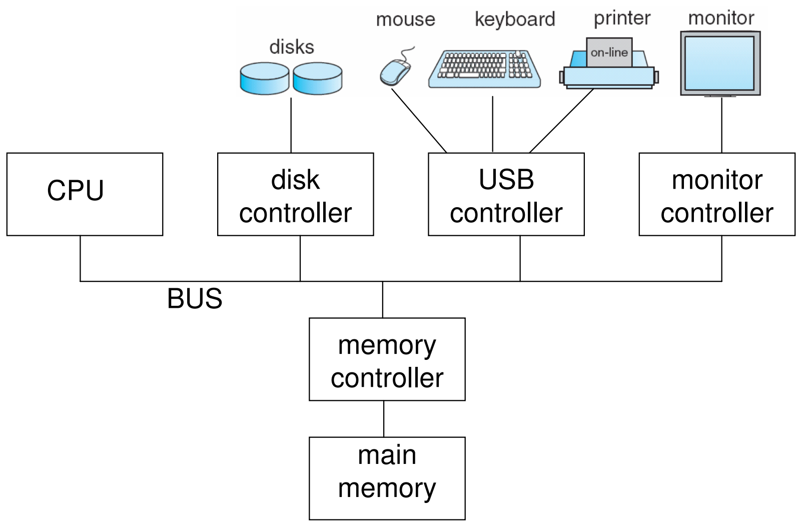

Figure 1.2 A modern computer system.

Most of the personal computers (PCs) have single CPU (also called **processor**). More powerful computers may have several CPUs. Such computers are usually referred to as **multiprocessor** systems. These computing devices are mainly used in a situation where one needs to compute a time intensive computation (for example, weather prediction), or when the result is needed very fast. Recently, as the price of PCs came down, we start seeing PCs with multiple processors.

The basic computing device is commonly referred to as **hardware**. The instructions to manipulate the data stored in the computing device are commonly referred to as **software**. A set of instructions to carry out a specific task is commonly referred to as a **program**.

1.2.3 Instructions

So how does a computer know what to compute? For example, how to add a set of 10 integers and print out the result. For this, the computer needs to have a program, which is usually written by a person (the **programmer**). You know by now that a program is a set of instructions. What we have not told you yet is what an instruction is. To do so, we need to get a closer look at the main memory and the CPUand the way they interact. What follows is a somewhat simplified view.

The main memory is simply an array of 4-bytes words (see Figure 1.3). The first word is residing in location "0" (or address "0"), the next word is residing in location "1", and so on.

The CPU has a number of registers that are the same size as a word. You can think of a register as "high-speed" memory word; that is, a register can be accessed and manipulated much faster than a memory word.

All instructions reside in main memory and are executed within the CPU. The execution of an instruction typically involves one or more of the CPU registers and one memory location. Here are some of the basic instructions one may find in a typical computer system:

- **load**—copy the content of a memory word to one of the CPU registers. For example, **load 212,3** will copy the content of memory location 212 to register 3.
- **store**—copy the content of one of the CPU registers to a memory word. For example, **store 2,4001** will copy the content of register 2 to memory location 4001.
- **add**—add the content of a memory word and the content of one of the CPU registers (say register 3) and store the new value into that register (register 3)
- **goto**—find the next instruction to be executed to be residing in a specific memory location.

Note that in general, an instruction consists of two parts: the **opcode**—what needs to be done (e.g., **load**. **add**), and the **operand**—the memory location specified in the instruction (for simplicity we ignore the registers specified in an instruction).

The functionality of the **load**, **store**, **add**, or **goto** machine instructions are provided by the hardware of the CPU. There are digital circuits for adding two numbers. This "adder" is built out of very simple logical "AND", "OR", and "NOT" functions. For instance, an AND circuit takes 2 bits as input values and produces the following output:

input 1	*input 2*	*output*
0	0	0
0	1	0
1	0	0
1	1	1

Only if both input values are "1" the output yields "1" too. This is why the circuit is called AND-circuit.

Recall that instructions and data are represented as a binary sequence. So the instruction **load 212,3** might be represented by the sequence:

11.........11

1.2.4 Program Execution

A computer program must be in main memory to be executed. The program consists of a finite sequence (say N) of instructions, which can be referred to as:

instruction 1, instruction 2, . . . , instruction N.

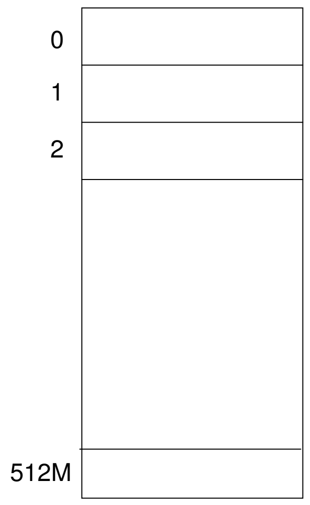

Figure 1.3 Simple view of main memory.

Each instruction resides in a single memory word, and the instructions are sequenced in memory one after the other. Accordingly, if the first instruction (instruction 1) resides in memory location 1001, then instruction 2 resides in memory location 1002, and so on.

A typical instruction−execution cycle, first fetches an instruction from memory and stores that instruction in the **instruction register**. The instruction is then decoded to determine what actions need taken (e.g., load). For example, in the case of a "load 3, 1002" instruction, the content of memory location 1002 is copied into register 3. In the case of a "goto 2000" instruction, the next instruction to be executed is the one residing in location 2000, not the one that is next in the instruction sequence.

Notice that the memory unit sees only a stream of memory addresses; it does not know how they are generated or what they are for (instructions or data). Accordingly, we can ignore *how* a memory address is generated by a program. We are interested only in the sequence of memory addresses generated by the running program.

1.2.5 Booting The System

For a computer to start running—for instance, when it is powered up or rebooted—it needs to have an initial program to run. This initial program, or **bootstrap program**, tends to be simple. Typically, it is stored in read-only memory (ROM) or electrically erasable programmable read-only memory (EEP-ROM), known by the general term **firmware**, within the computer hardware. It initializes all aspects of the system, from CPU registers to device controllers to memory contents.

1.3 Programming

When a person wishes to solve a particular problem using the computer, that person usually needs to write a computer program or use an already available program. A program instructs the the computer what needs to done. The language in which one writes computer programs is commonly referred to as a **programming language**.

There are numerous programming languages available No matter which language you use, the design of a program usually consists of three major steps:

1. **Developing an algorithm**
2. **Deciding on the data structures**
3. **Writing the actual code**

Below, we briefly outline the major activities that need to take place in each of the steps above.

1.3.1 Algorithms

An **algorithm** is a finite set of instructions for accomplishing some specific task. The instructions need not be computer-based instructions, and an algorithm need not be implemented as a computer program. Rather, it is a formal way of specifying a solution to a specific problem one wishes to solve.

To illustrate, suppose you are given an ordered set of N ($N \geq 2$) integers, and you want to find out the the smallest integer in the set. Let *IN* be the name of the set. One way of accomplishing this is to define the following algorithm, which consists of the following sequence of instructions:

1. Pick the first integer in the set *IN* and place it on the side.
2. Pick the next integer in the set *IN* and compare it with the integer on the side. If the "next" integer is smaller than the integer you placed on the side exchange the "next" integer with the one on the side.
3. If this is not the last integer in the set *IN* go to step (2).
4. If this is the last integer in the set *IN*, then the integer on the side is the smallest integer in the set *IN*.

We encourage the reader to create a set of 10 paper cards, each consisting of an arbitrary integer, and to "execute" the above algorithm, to ascertain that when the execution completes the integer on the side is indeed the smallest.

The above example illustrates several important properties of what an algorithm is:

- An algorithm consists of a finite sequence of steps. Some of these steps are quite simple (e.g., exchange a set of integers), some of these steps are decision-based (e.g., if this is not the last integer in the set *IN* ...), and some of these steps are iteration-based (e.g., repeat a sequence of steps until the algorithm completes).

- An algorithm usually has some data it operates on (e.g., the set *IN* and and the "integer on the side").
- An algorithm usually has some input (e.g., the set *IN*) and some output (e.g., the "integer on the side").

There are many different "correct" algorithms for solving the exact same problem. As we shall see in Section 2.7.2, not all algorithms for solving the same exact problem are created equal. Some may take less time to complete than others, may require less memory space than others, etc. Hence, it is not always sufficient to devise a correct algorithm; one may want also to make sure that it executes efficiently.

1.3.2 Data

Integral to a computer program is the presence of data. Each data element in a computer program, in its simplest form, is a sequence of bits that represents some information. Programmers, however, prefer to view the data not as sequence of bits, but as the actual information that the data represents.

As you will see throughout the text, there are a number of fundamental data types that all programming languages support, including:

- **integers**—both positive and negative integers.
- **real**—both positive and negative real numbers
- **characters**—alphanumeric character strings.

Thus a programmer can declare in his program that there will be 4 data items: 2 integers, 1 real, and 1 character.

1.3.3 Coding

An algorithm provides the blue print for creating a computer program. Once you have devised an algorithm and convinced yourself that it is correct, you are ready to start writing the corresponding program.

In Section 1.2 we had introduced the concepts of an "instruction" and a "sequence of instructions", which we call a program. The instructions we have discussed are low-level instructions. When they are represented as a sequence of binary digits, they are referred to as **machine-level** instructions. A computing device understands only machine-level instructions.

Writing programs in machine language is a tedious job. Indeed, very few people these days write programs in machine language. Human beings prefer to write code (programs) in a higher-level notation, which, at the vary least, is symbolic, rather than binary.

One of the simplest high-level notation is the **assembly language**, which closely resembles machine language, except that instead of specifying the operands and addresses as bits we specify them as symbols.

Although an assembly language makes it easier to write code in, it nevertheless, is still very low-level. Most of the programming today is done in much higher-level notation, which are commonly referred to as **high-level programming languages**.

There are numerous higher-level notations available for writing computer programs. The most common ones being:

- C
- C#
- C++
- Java
- FORTRAN
- COBOL

In this text we primarily cover C and Java.

There are other higher-level languages, which are more specialized. For example, SQL is a language specifically designed for writing database applications.

The program you write in either a high-level language or an assembly language is commonly referred to as **source code**. The source code is translated to its corresponding machine-level program by the **compiler**—for a high-level language, or an **assembler**—for an assembly language. The machine-level program is commonly referred to as **object code** (or **binary code**).

1.4 Computer-System Components

We now have a general idea what a computing device looks like and what it does. We also have a rough idea what programming is all about. We can now put all of these together and discuss the general organization of a computer system. A computer system can be divided roughly into five components: the **hardware,** the **operating system,** the **system programs, application programs,** and the **users** (Figure 1.4).

1.4.1 Hardware

The hardware—the CPU, the memory, and the input/output (I/O) devices —provides the basic computing resources for the system. The operating system controls and coordinates the use of the hardware among the various system, application, and user programs. The **system and programs**—such as compilers, word processors, spreadsheets, and web browsers—define the ways in which the computer resources are used to solve users' computing problems.

Until a decade ago there were very clear boundaries among the five components. Recently, however, these boundaries start being more blurry.

1.4.2 Operating Systems

Most users of a computing system do not interact directly with the hardware. Rather, they use the **operating system**, which is a program that manages the computer hardware. It acts as an intermediary between the computer user and the computer hardware.

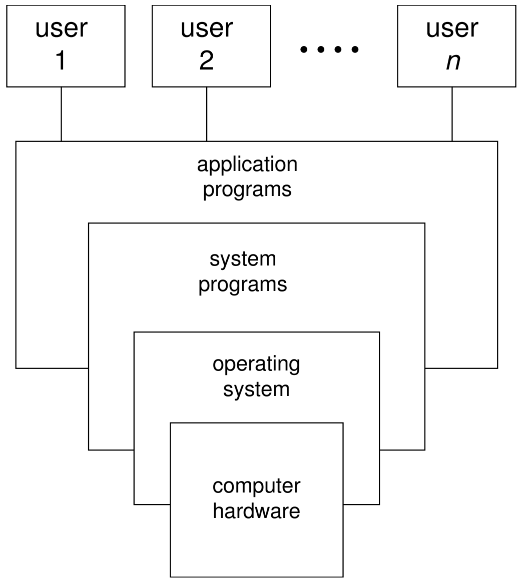

Figure 1.4 Abstract view of the components of a computer system.

As we saw before, we can view a computer system as consisting of hardware, software, and data. The operating system provides the means for proper management of these resources. An operating system is similar to a *government*. Like a government, it performs no useful function by itself. It simply provides an *environment* within which other programs can do useful work.

One of the most visible part of an operating system is the **file system**. A file is a collection of related information defined by its creator. Commonly, files represent programs (both source and object forms) and data. Data files may be numeric, alphabetic, alphanumeric, or binary. Files may be free-form (for example, text files), or they may be formatted rigidly (for example, fixed fields). The operating system implements the abstract concept of a file by managing mass storage media, such as disks, and the devices that control them.

The user's view of the computer varies according to the interface being used. Most computer users sit in front of a PC, consisting of a monitor, keyboard, mouse, and system unit. Such a system is designed for one user to monopolize its resources. The goal is to maximize the work (or play) that the user is performing. In this case, the operating system is designed mostly for **ease of use**, with some attention paid to performance and none paid to **resource utilization**—how various hardware and software resources are shared. Performance is, of course, important to the user; but rather than resource utilization, such systems are optimized for the single-user experience.

In other cases, a user sits at a terminal connected to a **super computer**—a computer that is very powerful and expensive and is shared among many users. Other users are accessing the same computer through other terminals. These users share resources and may exchange information. The operating system in such cases is designed to maximize resource utilization—to assure that all available CPU time, memory, and I/O are used efficiently and that no individual user takes more than her fair share.

Recently, many varieties of handheld computers have come into fashion. Most of these devices are standalone units for individual users. Some are connected to networks, either directly by wire or (more often) through wireless modems and networking. Because of power, speed, and interface limitations, they perform relatively few remote operations. Their operating systems are designed mostly for individual usability, but performance per amount of battery life is important as well.

Some computers have little or no user view. For example, embedded computers in home devices and automobiles may have numeric keypads and may turn indicator lights on or off to show status, but they and their operating systems are designed primarily to run without user intervention.

1.4.3 System Programs

Although the operating system shields the user from the low-level details of the hardware, it nevertheless does not provide sufficient tools to allow the users to interact with the computer system both efficiently and conveniently. For this purpose, a number of different system programs exist, including:

- **Assemblers**, to allow a user to write assembly language code. As we had indicated before, very few people write such code. It is usually done when there are no high-level languages available or when efficient coding is of upmost importance.
- **Compilers**, to allow a user to write code in some high-level language.
- **Text Editors**, to allow a user to create simple text documents.
- **Mail Systems**, to allow a user exchange e-mail messages with other users.
- **Web Browsers**, to allow a user to look at information available on the Internet and conduct some basic simple transactions (e.g., buy airline tickets).

There are some other classes of system programs, such as **database systems**. A database is a collection of interrelated data and a set of programs to access those data. The collection of data, usually referred to as the **database**, contains information relevant to an enterprise. The primary goal of such a database system is to provide a way to store and retrieve database information that is both *convenient* and *efficient*.

1.4.4 Application Programs

With the availability of hardware, operating system, and the various system programs outlined above, one can start building customized programs, which are sometime refereed to as **application programs**. It is difficult to provide a precise definition of what an application program is. Suffices to say that such a program uses many of the functionalities provided by the operating system and system programs to create new “applications”. Here are a couple of illustrative examples:

- **Maps**. There are many systems today that provide users on the web the ability to locate streets, restaurants, hotels, etc. Some of these systems allow you get directions from point A to point B. They can be combined with a GPS system to allow one to follow a route that they are driving on, etc.
- **Office**. One can think about the various "office applications" (e.g., Word, Powerpoint, pdf) as application programs.
- **Commercial systems** like R/3 from SAP that covers enterprise resource planning (ERP), production planning (PPS), human resources (HR), business intelligence (BI), and many more.
- **Computer games**. The various computer games that one can load from the web or buy in a store are application programs.

Interestingly, one can build application programs on top of another application program for some added value. For example, when Google provided their mapping system, within a short period of time people were building application programs on to of it for such purposes of real-estate (www.housingmaps.com), crime-reporting information (www.chicagocrime.org), etc.

1.4.5 Users

A primary goal of a computer system is to store data and allow the users to manipulate the data. There are four different types of computer-system users, differentiated by the way they expect to interact with the system. Different types of user interfaces have been designed for the different types of users.

- **Naive users** are unsophisticated users who interact with the system by invoking one of the application programs that have been written previously. For example, when a user wishes to withdraw $150 from an ATM machine, the machine invokes a program called *withdraw*. This program asks the user for account-number and password, and the the amount of money to be withdrawn. After verifying that the information is valid, the account information is updated and the $150 are dispensed.

  As another example, consider a user who wishes to find her account balance over the World Wide Web. Such a user may access a form, where she enters her account number. An application program at the Web server then retrieves the account balance, using the given account number, and passes this information back to the user.

  The typical user interface for naive users is a forms interface, where the user can fill in appropriate fields of the form. Naive users may also simply read *reports* generated from some application program.
- **Application programmers** are computer professionals who write application programs. Application programmers can choose from many tools to develop user interfaces. **Rapid application development (RAD)** tools are tools that enable an application programmer to construct forms and reports with minimal programming effort.
- **System Programmers** are computer professionals who write or modify operating system, database system, network system, compiler, etc. code.

- **System Administrators** are sophisticated users who are in charge of a specific computer system (or systems). Their function include the creation and deletion of user accounts, monitoring the performance of a system, writing specific system applications for their environment, etc. If you are the sole owner and user of a computer system, you are the di-facto system administrator of that machine.

1.5 Classes of Computer Systems

There are a number of different classes of computer systems. Some are general-purpose systems (e.g., PCs, mainframe, and supercomputers), while others are special-purpose systems whose functions are more limited and whose objective is to deal with limited computation domains.

We have already provided a basic coverage of general-purpose system. In this section we will present a general overview of two important classes of special-purpose systems—embedded systems and handheld systems.

1.5.1 Embedded Systems

Embedded computers are the most prevalent form of computers in existence. These devices are found everywhere, from car engines and manufacturing robots to VCRs and microwave ovens. They tend to have very specific tasks. The systems they run on are usually primitive, having little or no user interface. These system mostly spend their time monitoring and managing hardware devices, such as automobile engines and robotic arms.

These embedded systems vary considerably. Some use general-purpose hardware, running standard operating systems—such as UNIX—with special-purpose applications to implement the functionality. Others are hardware devices with a special-purpose embedded operating system providing just the functionality desired. Yet others are hardware devices with application-specific integrated circuits (**ASICs**) that perform their tasks without an operating system.

The use of embedded systems continues to expand. The power of these devices, both as standalone units and as members of networks and the Web, is sure to increase as well. Even now, entire houses can be computerized, so that a central computer—either a general-purpose computer or an embedded system—can control heating and lighting, alarm systems, and even coffee makers. Web access can enable a home owner to tell the house to heat up before she arrives home. Someday, the refrigerator may call the grocery store when it notices the milk is gone.

Embedded systems almost always are **real-time systems**. A real-time system is used when rigid time requirements have been placed on the operation of a processor or the flow of data; thus, it is often used as a control device in a dedicated application. Sensors bring data to the computer. The computer must analyze the data and possibly adjust controls to modify the sensor inputs. Systems that control scientific experiments, medical imaging systems, industrial control systems, and certain display systems are real-time systems. Some automobile-engine fuel-injection systems, home-appliance controllers, and weapon systems are also real-time systems.

A real-time system has well-defined, fixed time constraints. Processing *must* be done within the defined constraints, or the system will fail. For instance, it would not do for a robot arm to be instructed to halt *after* it had smashed into the car it was building. A real-time system functions correctly only if it returns the correct result within its time constraints. Contrast this system with a time-sharing system, where it is desirable (but not mandatory) to respond quickly, or a batch system, which may have no time constraints at all.

1.5.2 Handheld Systems

Handheld systems include personal digital assistants (PDAs), such as Palm and Pocket-PCs, and cellular telephones, many of which use special-purpose embedded systems. Developers of handheld systems and applications face many challenges, most of which are due to the limited size of such devices. For example, a PDA is typically about 5 inches in height and 3 inches in width, and it weighs less than one-half pound. Because of their size, most handheld devices have a small amount of memory (somewhere between 512 KB and 128 MB), slow processors, and small display screens.

The second issue of concern to developers of handheld devices is the speed of the processor used in the devices. Processors for most handheld devices run at a fraction of the speed of a processor in a PC. Faster processors require more power. To include a faster processor in a handheld device would require a larger battery, which would take up more space and would have to be replaced (or recharged) more frequently. Most handheld devices use smaller, slower processors that consume less power.

The last issue confronting program designers for handheld devices is I/O. A lack of physical space limits input methods to small keyboards, handwriting recognition, or small screen-based keyboards. The small display screens limit output options. Whereas a monitor for a home computer may measure up to 30 inches, the display for a handheld device is often no more than 3 inches square. Familiar tasks, such as reading e-mail and browsing web pages, must be condensed into smaller displays. One approach for displaying the content in web pages is **web clipping**, where only a small subset of a web page is delivered and displayed on the handheld device.

Generally, the limitations in the functionality of PDAs are balanced by their convenience and portability. Their use continues to expand as network connections become more available and other options, such as digital cameras and MP3 players, expand their utility.

1.6 Networking

Until now we have focused our attention on an environment where the various computer systems work in isolation. Although this environment provides a useful computation platform, a much more potent environment is one where the various computation devices can communicate with each other

1.6.1 Network Infrastructure

A **network**, in the simplest terms, is a communication path between two or more computing devices. The computing devices usually employ a **protocol** for data interchange. A protocol is a set of rules governing the orderly exchange of data among a number of computing devices. A protocol may be implemented in either hardware or software (or a combination of the two).

Networks are classified based on the distances between their computing devices. A **local-area network (LAN)** connects computers within a room, a floor, or a building. A **wide-area network (WAN)** usually links buildings, cities, or countries. A global company may have a WAN to connect its offices worldwide. These networks may run one protocol or several protocols. The continuing advent of new technologies brings about new forms of networks. For example, a **metropolitan-area network (MAN)** could link buildings within a city. BlueTooth and 802.11 devices use wireless technology to communicate over a distance of several feet, in essence creating a **small-area network** such as might be found in a home.

The media to carry networks are equally varied. They include copper wires, fiber strands, and wireless transmissions between satellites, microwave dishes, and radios. When computing devices are connected to cellular phones, they create a network. Even very short-range infrared communication can be used for networking. At a rudimentary level, whenever computers communicate, they use or create a network. These networks also vary in their performance and reliability.

1.6.2 Internet

The Web has become ubiquitous, leading to more access by a wider variety of devices than was dreamt of a few years ago. PCs are still the most prevalent access devices, with workstations, handheld PDAs, and even cell phones also providing access.

Web computing has increased the emphasis on networking. Devices that were not previously networked now include wired or wireless access. Devices that were networked now have faster network connectivity, provided by either improved networking technology, optimized network implementation code, or both.

Web technologies are stretching the boundaries of traditional computing. Companies establish **portals**, which provide web accessibility to their internal servers. **Network computers** are essentially terminals that understand web-based computing. Handheld computers can synchronize with PCs to allow very portable use of company information. Handheld PDAs can also connect to **wireless networks** to use the company's web portal (as well as the myriad of other web resources).

Today, network-connection speeds once available only at great cost are relatively inexpensive, giving home users speedy access to more data. These fast data connections are allowing home computers to serve up web pages and to run networks that include printers, client PCs, and servers. Many of the homes today even have **firewalls** to protect their computer data content from security breaches. Those firewalls cost thousands of dollars a few years ago and did not even exist a decade ago.

1.7 Security

The information and resources of a computer system must be protected from unauthorized access and malicious destruction or alteration of the data.

We say that a system is **secure** if its resources are used and accessed as intended under all circumstances. Unfortunately, total security cannot be achieved. Nonetheless, we must have mechanisms to make security breaches a rare occurrence, rather than the norm.

Security violations (or misuse) of the system can be categorized as intentional (malicious) or accidental. It is easier to protect against accidental misuse than against malicious misuse. For the most part, protection mechanisms are the core of protection from accidents.

Intruders (or attackers) are people or computer systems that are attempting to breach security. Attackers use several standard methods in their attempts to breach security. The most common is **masquerading**, in which one participant in a communication pretends to be someone else (another system or another person). By masquerading, attackers breach **authentication**, the correctness of identification; they can then gain access that they would not normally be allowed or escalate their privileges—obtain privileges to which they would not normally be entitled.

It is the job of **security** to defend a system from external and internal attacks. Such attacks spread across a huge range and include viruses and worms, denial-of-service attacks (which use all of a system's resources and so keep legitimate users out of the system), identity theft, and theft of service (unauthorized use of a system). Prevention of some of these attacks is consider an operating-system function on some systems, while others leave the prevention to policy or additional software.

Security require the system to be able to distinguish among all its users. Most operating systems maintain a list of user names and associated **user identifiers (user IDs)**. Furthermore, the system must provide mechanisms that allow the implementation of security features. Without the ability to authorize users, to control their access, and to log their activities, it would be impossible for an operating system to implement security measures or to run securely. Hardware protection features are needed to support an overall protection scheme.

Unfortunately, little in security is straightforward. As intruders exploit security vulnerabilities, security countermeasures are created and deployed. This causes intruders to become more sophisticated in their attacks.

1.8 So What is Computer Science?

We are now in a position to tell you what the field of computer science is all about. As we stated in the beginning of the chapter, computer science deals with the representation, organization, implementation, manipulation and communication of information in digital format. The information can be viewed in different level of abstractions, which are layered going from simple sequences of bits to complex cognitive processes. At each level of abstraction computer science has developed models to organize information effectively, called data structures, to process the information efficiently, called algorithms, and to communicate among the various levels, called protocols.

As you will see, there is rich body of knowledge concerning algorithms, data structures, and protocols. This body of knowledge had been applied to many different application domains. These form the foundation of the field of computer science.

1.8.1 Theory

Theoretical computer science deals with the study of the fundamental principles underlying computation. Some of the major issues are:

- **Fundamental Algorithms**. There are a number of basic algorithms that are used in many different applications. These algorithms are of significant importance and have been carefully studied an analyzed.
- **Fundamental Data Structures**. Just like fundamental algorithms, there are a number of basic data structures that are used in many different applications. These data structure are of great importance and have been carefully studied an analyzed.
- **Decidability**. Can a particular task be actually computed? That is, does there exist an algorithm to accomplish this task.
- **Complexity**. For a given algorithm, how costly is it to compute in terms of both time and space (time/space tradeoff).
- **Approximation**. If a particular algorithm is very costly to compute, can one devise an algorithm that is much less costly to compute that provides a "close" approximation of the original algorithm (time/accuracy tradeoff).
- **Parallel computation**. Current-day computer systems allow multiple programs to be loaded into memory and be executed concurrently. The individual units of execution are called **processes**. A process is a program in execution, and is the unit of work in a modern computing system. A fundamental issue concerning concurrent execution is process communication and coordination, which we cover in Chapter 14.
- **Distributed computation**. The availability of computer networks provides us with capability to distribute a single computation among a number of different computing devices, which are geographically distributed. This capability allows us to speed up a computation and to get added reliability. This, however, introduces interesting new challenges in terms of both communication and coordination, which we will cover in Chapter 15.

1.8.2 Systems

Many of the fundamental algorithms and data structures are used directly or are augmented to create new ones that are suitable for particular system applications.

- **Operating Systems**
- **Networking Systems**

- **Database Systems** A database is a collection of interrelated data and a set of programs to access those data. Database systems are designed to manage large bodies of information. Management of data involves both defining structures for storage of information and providing mechanisms for the manipulation of information. In addition, the database system must ensure the safety of the information stored, despite system crashes or attempts at unauthorized access. If data are to be shared among several users, the system must avoid possible anomalous results. There is a large body of concepts and techniques for managing data. Some of these concepts and techniques are covered in this text.
- **Programming languages and compilers**

1.8.3 Applications

The fundamental algorithms and data structures developed by the theoretical community are used directly or are augmented by the system and application community to create new ones that are suitable.

As you will see, there is rich body of knowledge concerning both data structures and algorithms. These form the foundation for variety of subfields of computer science, including:

It needs to deal with application type issues:

- **Scientific Computing**
- **Artificial Intelligence**
- **Graphics**

1.8.4 Software Engineering

Software engineering deals with methods to analyze, design and realize software. In addition, like other engineering disciplines, cost and reliability aspects are considered.

As a little side note, the term itself is under debate. Some people argue that we do not have yet the rigor and proven processes to call the development of software “engineering”. Pete McBreen pleas for the name “craftsmanship” to emphasize the skill of the developer instead of the “manufacturing” process.

1.8.5 Tying it All Together

The five subfields of computer science (hardware, theory, systems, applications, software engineering) are by no mean disjoint. The results produced in one subfield are sometimes used directly by another subfield. In other cases, the results need to be augmented to fit a particular need. This symbiotic relation has been going on since the inception of the field computer science and is extremely beneficial to all subfields.

As a final note, it is worth while mentioning that computer science is a unique field in that it is **technology driven**. When a new digital-based technology come along it usually requires the generation of new research in computer science. This new research, in turn, results in the creation of new technologies.

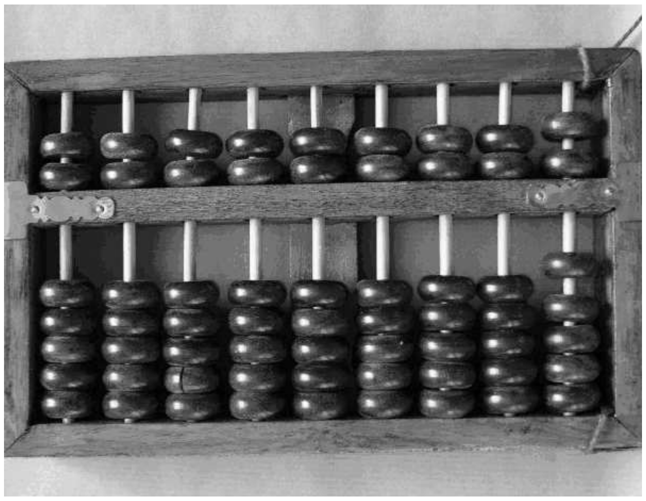

Figure 1.5 An ancient abacus replica

1.9 History of Computing

Computing has fascinated mankind at all times. It needs a “sense for numbers” which not only humans possess but also some animals like apes, elephants, and birds. But only man has developed an abstract number system to count objects and do computations with numbers.

Computing was long time considered an art and it was really complicated to compute the sum of two numbers in Roman symbols, Egyptian or Sumerian hieroglyphs. The reason for the complication was that these systems had no symbol for *zero* and therefore could not develop a positional system for the digits. Egyptians, Babylonians, and Greeks developed tables for multiplications and trigonometric computations. These have been used by government officials, scholars, geometers, and sailors. The abacus was a first attempt to do calculations with a simple machine. It consists of a frame with beads sliding on wires or slats (see figure 1.5). Addition and subtraction was reduced to counting beads.

The discovery of the number *zero* may be esteemed by the fact that this happened only three times to mankind: the Babylonian scholars, the Maya astronomers, and the Indian mathematicians.

It took until 1524 when Adam Riese wrote a book on computation to spread the Indian decimal system in Europe. From that time on, everybody could learn how to add two numbers because the instruction consisted of step-by-step actions to follow. The idea of “mechanizable steps” was called *algorithm* to honor the Persian mathematician *Al-Khwarizmi*.

There have been many attempts to build machines for doing complex calculations with the Indian decimal system. Known in Europe are Wilhelm Schickard (1592-1635), Blaise Pascal (1623-1662), and Gottfired Leibniz (1646-1716). But their success was limited due to fine mechanical problems. And, all these machines were missing two important elements of the modern computers. They had no memory and were not programmable. The *algorithms* were “built in” by wheals, transmission, and gears.

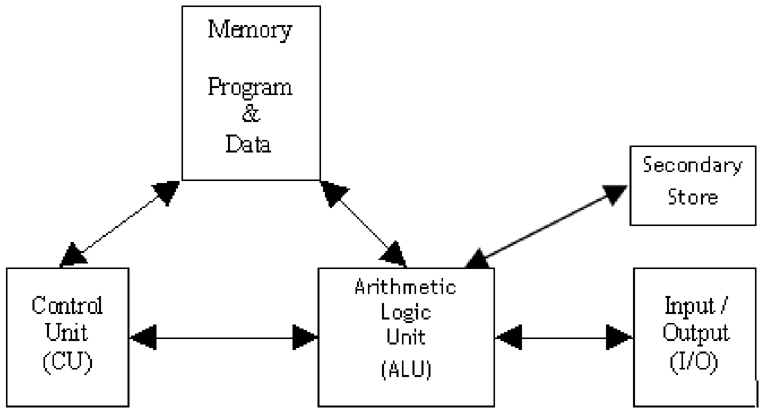

Figure 1.6 Block diagram of the von Neumann computer

1.9.1 The Ancestors of the Computer

The first programmable computer was the automatic loom by Joseph-Marie Jacquard. The weaving pattern was stored as holes in wooden cards. The cards were connected to form a loop so that the weaving pattern could be repeated as often as desired. To weave a different pattern it was only necessary to exchange the cards.

The most important precursor of modern computer scientists was the English mathematician *Charles Babbage*. He developed the idea of a programmable calculation machine, capable of doing addition, subtraction, multiplication, and division of numbers up to six digits long. The design of his *analytical engine* consisted of a *mill* to perform arithmetic computations, a data *store*, an *operation* unit to process the instructions coded on the punched cards, and an *output* unit to print the results.

Ada Augusta Byron, Countess of Lovelace, translated a French description of the analytical engine, annotated it extensively, and added a detailed description how to compute Bernoulli numbers with this machine. This description is recognized by historians as the world's first computer program and gave reason to recognize her to as the first programmer.

Unfortunately Babbage's invention failed to work due to the mechanical challenge. In recent years a fully functional replica of the analytical engine was built by using a model construction kit.

More than a hundred years later *John von Neumann* proposed a similar model for a modern computing system (see figure 1.6). The mill corresponds to the arithmetic and logic unit (ALU), the store is now called memory, the operator's functions are now covered by the control unit, and the output became the input/output unit.

1.9.2 The Birth of Computers

A real progress to computing machines came in 1934 by the German engineer *Konrad Zuse* who had the idea of using the binary numbering system for computing. His first computer the Z1 was still a mechanical construction, but

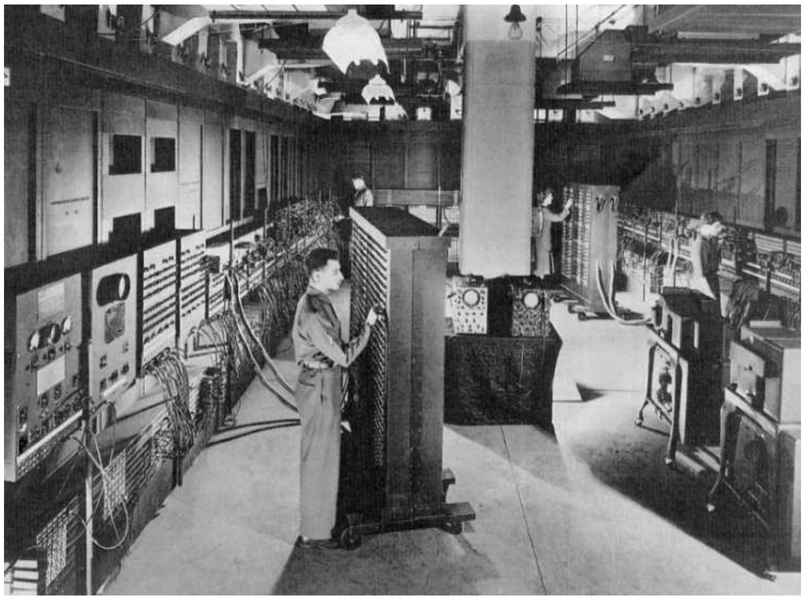

Figure 1.7 The ENIAC Computer

used the binary system for calculations. The Z3 completed in 1941 was the first operational binary computer using electromechanical technology (relays).

In the USA *Howard Aiken* developed at Harvard an electromechanical computer, the *Mark I* , based on the decimal system that was completed in 1944. It could store 72 numbers, made three additions per second and could do a multiplication in 4 seconds.

The ENIAC (Electronic Numerical Integrator and Computer) constructed by *Eckert* and *Mauchley* did the same multiplication in 1/300 of a second due to the use of vacuum tubes. One serious technical obstacle was the unreliability of the vacuum tubes. Using nearly 18000 tubes for the ENIAC resulted in a mean time between failure (MTBF) of 6 to 12 hours according to different sources.

The ENIAC consumed 174 kW and took up 167 m^2 of space (figure 1.7). The longest reported period of operation without a failure was approximately 5 days. It was programmed by using wires - called patch cables - like in an early telephone switch center. It was the idea of *John von Neumann* who proposed to store the program like data. This was truly revolutionary because it allowed for easy reprogramming the computer by just changing the stored program code.

The computing model that *von Neumann* proposed consisted of an arithmetic and logic unit (ALU), a control unit (CU), a memory (M), an input/output unit (I/O), and a secondary store. The data and instructions were loaded from the secondary store into memory. The first instruction was transfered from memory into the control unit and executed. The execution involved transferring data from memory to the arithmetic and logic unit (ALU) and storing intermediate results back into memory. Then, the next instruction was transfered from memory to the control unit overriding the previous instruction.

After the sequence of instructions (today called *program*) had executed completely, the result was transfered to the output device.

At about the same time different computer architectures and technologies evolved in the USA (ABC, Mark I), Great Britain (Colossus), and Germany (Z3).

1.9.3 The Modern Computer

The first commercial computer, the *UNIVAC I* (Universal Automatic Computer) was build 1951 by Remington Rand and was sold to the US Census Bureau. Shortly after that, IBM built its first computer, the *IBM 701* which was the first one of the successful IBM 700/7000 series computers.

The computers using either relays or vacuum tubes are called **first generation** computers. Their computing power reached 1000 additions per second.

The **second generation** has started in the late fifties and lasted to the end of the sixties. The unreliable tubes were replaced by transistors and as storage medium core memory and drum stores were used. The computing power grew to 100000 additions per second (10 μs/addition). The use of semiconductors reduced the energy consumption considerably and solved the reliability problems of the first generation. The miniaturization driven by the use of transistors continues since that time until present.

A real boost for smaller and faster computers brought the era of *integrated circuits* (IC). The **third generation** of computers is marked by these ICs and lasted from about 1965 to 1975.

The **fourth generation** used “one-chip” microcomputers that integrated the ALU, the control unit (CU), and some memory (here called *cache*). The first microcomputer, the *Altair 8800* had only the size of a todays desktop computer. This was the advent of personal computing. Word processing, spread sheet computing, and other productivity tools boosted the distribution of *personal computers* (PC) into every office. The increasing power of the *central processing unit* (CPU), containing the ALU and the CU, allowed graphical user interfaces that made it easier for casual users to work with a PC.

During the fourth generation (1975 to 1985) computers started to get connected and formed computer networks which enabled users to exchange messages (e-mail) via computers. The *Internet* grew out of military and university networks and is now mainly used for the *world wide web* (WWW), IP-telephone, video streaming, file transfer and many other web based services.

Some people did stop counting computer generations as the development forked into different trends and technologies. The Japanese **fifth generation** project on massively parallel processing started todays computer generation. The goals were to make computers more human like (speech recognition, autonomous systems including robotics, artificial intelligence, high performance computing).

Todays computer generation is characterized by ultra-integrated circuits, massively parallel computers, miniaturized mobile computers, and ubiquitous computing using embedded systems.

In only 60 years the development of computers have progressed from electromechanical devices having only a hundred cells of memory with “wired” programs and a few operations per second to multicore laptops with gigabytes of memory and high resolution flat panel screens instead of punched cards and teletype writer. The development of speed, memory, and cost over the last

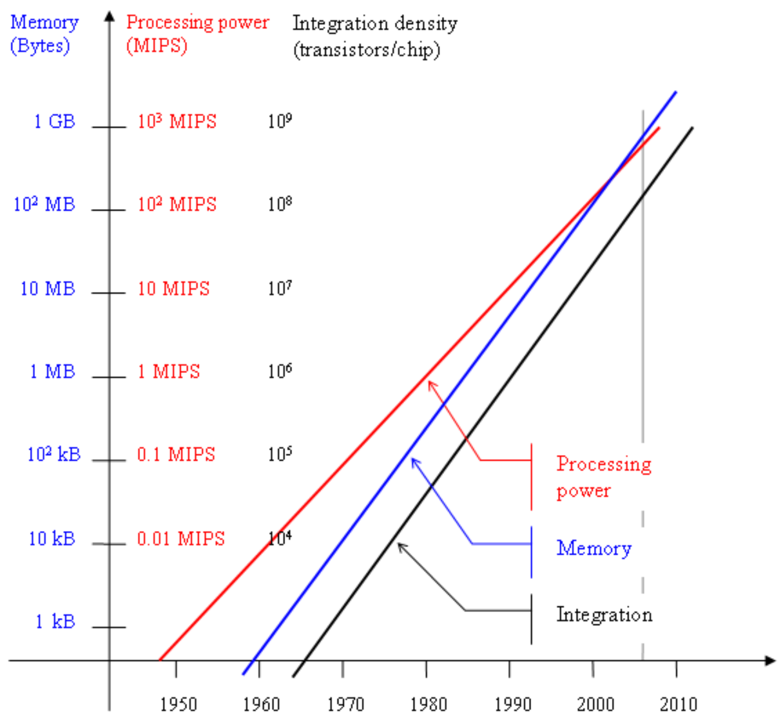

Figure 1.8 Moore's law

decades is shown by figure 1.8. The increase of speed and memory proves a statement by *Gordon Moore*, co-founder of Intel, who predicted that the integration density of IC will double every two years (**Moore's Law**).

Not only the hardware developed during the last 60 years but also software engineering made immense progress. The algorithms of the first generation computers had to be coded mainly in machine language. The next step was the assembler language which replaced binary code by mnemonic code like INCR for an increment instruction and symbolic addresses like *index* instead of 1010011. Assembler programming was wide spread during the second generation.

The first "high level" languages were FORTRAN (Formula Translation, 1954) and COBOL (Common Business Oriented Language, 1960) developed during the fifties. Their real success began with the processing power of the third generation computers. New programming paradigms arose in the seventieth like functional and object oriented programming. The most influencial functional language LISP (List Processing) was developed by *McCarty* (1958) at the MIT and Smalltalk (1972-1980) as object oriented language was developed at the Xerox Palo Alto Research Center (PARC) by a group of scientist led by *Alain Kay* and *Adele Goldberg*.

The idea of reusable software parts (modules) by *David Parnas* (1972) boosted software productivity. Todays progress is less in inventing new languages than to improve software architecture and building framworks like application servers to provide a highly abstract execution environment for modules having trendy names like servlet, enterprise Java beans, plug-in, or extension.

1.10 Summary

- Computer science deals with the representation, implementation, manipulation and communication of information in digital format. The information can be viewed in different level of abstractions, which are layered going from simple sequences of bits to complex cognitive processes.
- Computer science has developed models to organize information effectively, called data structures, to process the information efficiently, called algorithms, and to communicate among the various levels, called protocols.
- A digital computer is a general-purpose computing device. It can manipulate numbers (integers and real), symbols (alphanumeric as well as special symbols like ".", "$", etc.), audio signals, and video signals.
- A digital computer is based on a very simple scheme—both the hardware and software are digital-based. That is, the environment they are operating in assumes that the world consists of only two states, commonly referred to as **binary** and represented by the symbols "0" and "1".
- A digital computer consists of four basic components. **Main memory** (the place where the data and instructions to manipulate the data are stored), **CPU** (the control unit that executes the instructions), **I/O controllers** (the control units for the various peripheral devices), and **bus** (the communication structure for connecting the main memory, CPU, and the peripheral devices).
- A computer program is a sequence of machine-level instructions that must reside in main memory and are executed within the CPU.
- A program is written in either a high-level language or an assembly language and is commonly referred to as **source code**. The source code is translated to its corresponding machine-level program by the **compiler** —for a high-level language, or an **assembler**—for an assembly language. The machine-level program is commonly referred to as **object code** (or **binary code**).
- A **network**, is a communication path between two or more computing devices. The computing devices usually employ a **protocol** for data interchange. A protocol is a set of rules governing the orderly exchange of data among a number of computing devices. A protocol may be implemented in either hardware or software (or a combination of the two).
- The information and resources of a computer system must be protected from unauthorized access and malicious destruction/alteration of the data. This function is carried out by the **security system**.

Review Terms

- Digital computer
- Digital data
  - Bit
  - Byte
  - Word
- Main memory
- Random-access memory (RAM)
- Central Processing Unit (CPU)
- I/O controllers
- Communication bus
- Hardware
- Software
- Machine-level instructions
  - Opcode
  - Operand
- Computer programs
  - Source code
  - Object code
  - Binary code
- Algorithm
- Programming language
- Assembly language
- System programs
  - Assembler
  - Compiler
  - Operating systems
  - Database systems
- Application programs
- File system
- Real-time systems
- Handheld systems
- Networks
  - Protocols
  - local-area network (LAN)
  - wide-area network (WAN)
- Internet
- Security
- Theory
  - Decidability
  - Complexity
  - Approximation
  - Parallel computation
  - Distributed computation
- Applications
  - Scientific Computing
  - Artificial Intelligence
  - Graphics

Exercises

1.1 What is an algorithm?

1.2 The binary system is similar to the decimal system except that the number of symbols are two ("0" and "1") rather than ten ("0", "1", . . . , "9"). Thus, the integers are represented in a binary system as:

- integer 0—"0"
- integer 1—"1"
- integer 2—"10"
- integer 3—"11"

- etc.

How would the integers:

6, 10, 15, 28, 32

be represented in a binary system'?

1.3

1.4

1.5

1.6

Bibliographical Notes

Brookshear [2003] provides an overview of computer science in general. Other texts aimed at a low-level introduction of computer science include Biermann and Ramm [2002] and Snyder [2004].

There are many general textbooks covering operating systems, including Silberschatz et al. [2005], Stallings [2000b], Nutt [2004] and Tanenbaum [2001].

Textbooks covering database systems include Silberschatz et al. [2006], Date [2003], Elmasri and Navathe [2003], O'Neil and O'Neil [2000], Ramakrishnan and Gehrke [2002], Garcia-Molina et al. [2001], and Ullman [1988].

Hamacher et al. [2002] describes computer organization. Hennessy and Patterson [2002] provide coverage of I/O systems and buses, and of system architecture in general.

Kurose and Ross [2005], Tanenbaum [2003], Peterson and Davie [1996], and Halsall [1992] provide general overviews of computer networks. Fortier [1989] presents a detailed discussion of networking hardware and software.

Wolf [2003] discusses recent developments in developing embedded systems. Issues related to handheld devices can be found in Myers and Beigl [2003] and Di Pietro and Mancini [2003].

General discussions concerning security are given by Pfleeger and Pfleeger [2003], Tanenbaum 2003, and Russell and Gangemi [1991]. Also of general interest is the text by Lobel [1986].