Welcome to AP Computer Science (using JAVA)!

**Note:** There are two pages to this document, detailing two required summer assignments.

- Entering students need to be familiar with the application development process (specification, pseudocode, design, code, test, debug, maintenance), as well as with the basics of computer history, hardware and software. In addition you need to be versed on the ethical responsibilities of computer professionals and programming languages in general.
- As part of this class you must complete the following two assignments. They are both requirements of the AP Computer Science curriculum.

**Part I Ethics and Responsibilities in the Computing Professional:**
As a computer professional, you have ethical responsibilities to yourself, to your school or corporation, and to the public at large. There are many ethical questions about which you need to be informed, and concerned, including:

- Software Piracy, Copyright Laws, Privacy of Data, Responsible Use of Computer Resources
- System Reliability, Legal Issues and Intellectual Property, Social and Ethical ramifications of computer use
- Impact of applications using databases, particularly over the Internet on an individual’s right to privacy
- Economic and legal impact of viruses and other malicious attacks on computer systems.
- Intellectual property rights of writers, musicians and computer programmers and fair use of intellectual property

**Part I Assignment Details**
You are to write a three to five page paper, in MLA format (see [https://owl.english.purdue.edu/owl/resource/747/01/](https://owl.english.purdue.edu/owl/resource/747/01/) if you are not sure about proper formatting), addressing all of these topics. You must cite at least three sources, at least one of which is from a book or magazine.*

- On each topic, you should describe the problem that is posed, and provide a logical discussion on it, including current and pending laws regarding the topics
- On each topic you should provide your own opinions, including things like answering the question, “Should websites be required to charge for downloaded MP3 files, or should music be available for free to all who request it?” Why or why not?
- Create a brief PowerPoint or website presentation defending your positions that covers at least one of the topics that you researched.

All papers are due the second week of class in September. You will give a brief presentation (5 minutes) to the class. This will count as your first major grade.

**NOTE:** Barnes and Nobles has the user-friendly policy where they allow you to read their books in house. They have a large department containing computer books/magazines. That might be a cost-effective, efficient way to do your research (The Association of Computing Machines has a website [www.acm.org](http://www.acm.org) with information regarding this topic).

The public library ([http://www.gptx.org/index.aspx?page=171](http://www.gptx.org/index.aspx?page=171) for more information) is also open during the summer and you may request books not on site through interlibrary loan.

**Part II Introduction to Computers – A review:**
As an AP Computer Science student it is expected that you have a basic understanding of computer terminology. This is reviewed in Chapter 1 of our textbook, which will be provided to you for reference. Topics include:

- What is programming?, The anatomy of a computer: Becoming Familiar with your Computer
- Translating Human-Readable Programs to Machine Code; the Java Programming Language
- Compiling a Simple Program; Errors; the compilation Process.

**Part II Assignment Details**
- The chapter contains a series of Self-Check exercises (1-8 & 1-18) which you must be prepared to answer.
- A word document containing the answers to all 26 questions must be handed in for grading on the second day of class. This will count as your second homework grade.
- A test on this chapter will take place during the second week of class.
Some Technical Information:

- After you check out your laptop for next year, please see Edmodo (Group Code: 59ysbh) for instructions on installing the software we will use throughout the year. I will check for successful install on the first day of class and this will be your first daily grade of the year.
- Our textbook TBA. We will also use supplemental materials supplied by the teacher.
- Additional information regarding AP Computer Science can be found at the College Board website: http://www.collegeboard.com/student/testing/ap/sub_compscia.html?compscia

Bonus Summer Assignment:

- Before, during, or after you complete the real summer assignment – check out these fun “games”.
  - Lightbot - http://armorgames.com/play/2205/light-bot or find on iTunes
An Introduction to Hardware, Software, and the Internet

1.1 Prologue 1-2
1.2 Hardware Overview 1-5
   1.2.1 The CPU 1-5
   1.2.2 Memory 1-6
   1.2.3 Secondary Storage Devices 1-7
   1.2.4 Input and Output Devices 1-7
1.3 Software Overview 1-9
1.4 What Do Software Engineers Do? 1-10
1.5 Representation of Information in Computer Memory 1-14
   1.5.1 Numbers 1-15
   1.5.2 Characters 1-19
1.6 The Internet 1-21
1.7 Networking Basics 1-24
1.8 Summary 1-25
Exercises ✉
1.1 Prologue

The most important piece of a typical computer is the *Central Processing Unit* or *CPU* [1]. In a personal computer, the CPU is a microprocessor made from a tiny chip of silicon, sometimes as small as half an inch square. Immensely precise manufacturing processes etch a huge number of semiconductor devices, called *transistors*, into the silicon wafer. Each transistor is a microscopic digital switch and together they control, with perfect precision, billions of signals — little spikes of electricity — that arise and disappear every second. The size of the spikes doesn’t matter, only their presence or absence. The transistors in the CPU recognize only two states of a signal, “on” or “off,” “high” or “low,” “1” or “0,” “true” or “false.” This is called *digital electronics* (as opposed to *analog electronics* where the actual amplitudes of signals carry information).

The transistors on a chip combine to form logical devices called *gates*. Gates implement *Boolean* operations (named after the British mathematician George Boole, 1815-1864 [1] who studied the properties of logical relations). For example, an *AND* gate takes two inputs and combines them into one output signal. The output is set to “true” if both the first and the second input are “true,” and to “false” otherwise (Figure 1-1-a). In an *OR* gate, the output is set to “true” if either the first or the second (or both) inputs are true (Figure 1-1-b). A *NOT* gate takes one input and sets the output to its opposite (Figure 1-1-c). Note the special shapes used to denote each type of gate.

These three basic types of gates can be combined to make other Boolean operations and logical circuits. Figure 1-2, for example, shows how you can combine AND, OR, and NOT gates to make an *XOR* (“eXclusive OR”) operation. This operation sets the output to “true” if exactly one of its two inputs is “true.” In the late 1940s, John von Neumann, [1] a great mathematician and one of the founding fathers of computer technology, showed that all arithmetic operations can be reduced to AND, OR, and NOT logical operations.
The microprocessor is protected by a small ceramic case mounted on a PC board (Printed Circuit board) called the motherboard [1, 2]. Also on the motherboard are memory chips. The computer memory is a uniform pool of storage units called bits. A bit stores the smallest possible unit of information: “on” or “off,” “1” or “0.” For practical reasons, bits are grouped into groups of eight, called bytes.
One byte is eight bits.

There is no other structure to memory: the same memory is used to store numbers and letters and sounds and images and programs. All these things must be encoded, one way or another, in sequences of 0s and 1s. A typical personal computer made in the year 2010 had 2 to 4 “gigs” (gigabytes; 1 gigabyte is \(2^{30} \approx 10^9\) bytes) of RAM (Random-Access Memory) packed in a few SIMMs (Single In-Line Memory Modules).

The CPU interprets and carries out computer programs, or sequences of instructions stored in the memory. The CPU fetches the next instruction, interprets its operation code, and performs the appropriate operation. There are instructions for arithmetic and logical operations, for copying bytes from one location to another, and for changing the order of execution of instructions. The instructions are executed in sequence unless a particular instruction tells the CPU to “jump” to another place in the program. Conditional branching instructions tell the CPU to continue with the next instruction or jump to another place depending on the result of the previous operation.

Besides the CPU, a general-purpose computer system also includes peripheral devices [1] that provide input and output and secondary mass storage. In a laptop, the “peripheral” devices are no longer quite so peripheral: a keyboard, a display, a hard drive, a DVD drive, a wireless network adapter, a web cam (camera), a touch pad, a microphone, and speakers are all built into one portable unit.

CPU, memory, peripherals — all of this is called hardware. The most popular hardware platforms these days are smartphones, tablet computers, video game consoles, laptops [1]. Other platforms range from desktop personal computers to huge servers and supercomputers [1]. But to make it useful, to bring life into it, you need programs, software. Computer programs are also miracles of engineering, but of a different kind: software engineering. They are not cast in iron, nor even silicon, but in intangible texts that can be analyzed, modified, translated from one computer language into another, copied into various media, transmitted over networks, or lost forever. Software is to a computer as tunes are to a band: the best musicians will be silent if they don’t have music to play.

Take this amazing device with its software and connect it to the Internet, a network of billions of computers of all kinds connected to each other via communication lines of all kinds and running programs of all kinds, and you end up with a whole new world. Welcome to cyberspace!
1.2 Hardware Overview

1.2.1 The CPU

What a CPU can do is defined by its instruction set and internal *registers*. The registers are built-in memory cells that hold operands, memory addresses, and intermediate results. Some of the registers are accessible to the programmer. The instruction set includes instructions for loading CPU registers from memory and storing their values into memory, for logical and arithmetic operations, and for altering the sequence of operations.

Every computer has an internal “clock” that generates electrical pulses at a fixed frequency. All CPU operations and their component steps are synchronized with the clock’s pulses; their duration is measured in *clock cycles*. The CPU’s speed depends on the frequency of the clock. The Intel 8088 microprocessor in the original IBM Personal Computer (1981), for example, ran at 4.77 MHz (megahertz, or million pulses per second). Thirty years and dozens of computer generations later, CPU chips are pushing into the 3 GHz range (1 GHz, gigahertz, is equal to 1,000 MHz) \(^1\).

A microprocessor CPU connects to memory and to other devices through a set of parallel lines (wires printed on the motherboard) controlled by digital electronics, called a *bus* \(^1\). A CPU uses a separate *address bus* for specifying memory addresses and a *data bus* for reading and writing memory values. Besides the internal clock speed, the computer’s overall performance depends on the speed of the bus transfers and the width of the bus. The Mac 512k computer \(^1\), introduced in September of 1984, had the Motorola MC68000 8-MHz CPU and a 16-bit bus running at 8 MHz, so it could carry 16 bits of data concurrently from memory to the CPU roughly at the same speed as the CPU could handle the data. The MacBook released by Apple Computer Inc. in 2010 has a 64-bit bus running at around 1 GHz.
1.2.2 Memory

Each byte of memory has a unique address that can be used to fetch the value stored in the byte or write a new value into it. A CPU does not have to read or write memory bytes sequentially; bytes can be accessed in any arbitrary order. This is why computer memory is called random-access memory or RAM. This is similar to a CD where you can choose any track to play, as opposed to a tape that has to be played in sequence.

Table 1-1 shows units used to describe memory size. (Powers of 2 have a special significance in computer technology for a reason that will become clear shortly.)

<table>
<thead>
<tr>
<th>Name</th>
<th>Abbrev.</th>
<th>Bytes</th>
<th>Equals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kilobyte</td>
<td>KB</td>
<td>$2^{10} = 1,024$</td>
<td></td>
</tr>
<tr>
<td>Megabyte</td>
<td>MB</td>
<td>$2^{20} = 1,048,576$</td>
<td>1024 KB</td>
</tr>
<tr>
<td>Gigabyte</td>
<td>GB</td>
<td>$2^{30} = 1,073,741,824 \approx 10^9$</td>
<td>1024 MB</td>
</tr>
<tr>
<td>Terabyte</td>
<td>TB</td>
<td>$2^{40} \approx 10^{12}$</td>
<td>1024 GB</td>
</tr>
<tr>
<td>Petabyte</td>
<td>PB</td>
<td>$2^{50} \approx 10^{15}$</td>
<td>1024 TB</td>
</tr>
</tbody>
</table>

Table 1-1. Memory units

In the early days, designers of personal computers thought 64KB of RAM would suffice for the foreseeable future. An additional hardware mechanism, the segment registers, had to be added to the later versions of Intel’s microprocessors to access a larger memory space, up to one megabyte, while maintaining compatibility with the old programs. But the one megabyte limit very quickly proved inadequate too. A 32-bit memory address bus allows programs to directly address four gigabytes (GB) of memory. This is becoming a bottleneck as computers beginning to appear with 8GB of RAM.

A small part of the computer memory is permanent non-erasable memory, known as read-only memory or ROM. ROM contains, among other things, the initialization code that boots up the operating system (that is, loads into memory the boot record or initialization code from the disk and passes control to it).
An executable program has to be loaded from a hard disk or another peripheral device (or from the Internet) into RAM before it can run.

ROM solves the “first-program” dilemma — some program must already be running to load any other program into memory. The operating system is a program that has the job of loading and executing other programs. In a personal computer, ROM also contains the computer configuration program and hardware diagnostic programs that check various computer components. The ROM BIOS (Basic Input Output System) contains programs for controlling the keyboard, display, disk drives, and other devices. A special small memory — EPROM (Erasable Programmable ROM) — preserves system configuration data when the power is off.

### 1.2.3 Secondary Storage Devices

A computer’s RAM has only limited space, and its contents are wiped out when the power is turned off. All the programs and data in a computer system have to be stored in secondary mass storage. The auxiliary storage devices include hard disks, CD-ROM drives, USB flash drives, and other devices. A hard disk can hold hundreds of gigabytes and even terabytes. A CD-ROM can hold more than 700 MB (or up to 2 or 3 GB with data compression). Access to data on these devices is much slower than access to RAM.

The operating system software organizes the data in secondary storage into files. A file may contain a set of related data, a program, a document, an image, and so on; it has a unique name. The operating system maintains a directory of file names, locations, sizes, dates and times of the last updates, and other attributes.

Thus a “file” is not a hardware but rather a software concept.

When a program is running, it can read and write data directly to and from files stored on secondary storage devices.

### 1.2.4 Input and Output Devices

A personal computer receives user input through the keyboard and displays the output on the computer display (also called the monitor). In many programs the input is echoed on the screen as you type, creating the illusion that the keyboard is directly connected to the monitor. In fact these are two entirely different devices connected only through the CPU and the currently running program. The keyboard sends to the program digital codes that represent the pressed keys. The program captures these
codes and takes appropriate actions, which may include displaying corresponding characters on the screen.

The screen is controlled by a video adapter and displays the contents of special video memory in the adapter, called VRAM. VRAM is addressable by the CPU and may contain codes, colors and attributes of characters when running in text modes, or colors or intensities of individual pixels (“picture elements”) in graphics modes. The original IBM PC ran mostly in the text mode. You don’t see the text mode much any more — everything is in graphics now, except diagnostic and setup programs.

A mainframe computer [1, 2] (a very large multi-user computer) may have hundreds of terminals attached to it. The terminals send keystrokes and receive commands and character codes from the computer through digital transmission lines.

Printers, plotters, digitizing tablets, scanners, and other devices receive commands and data from the computer in digital form and may send data or control codes back to the computer according to a specific communications protocol.

Network adapters and modems are essential for getting your computer connected to other computers. Network adapters and cables are used to connect several computers into a LAN (Local Area Network), which in turn may be connected to the Internet.

Special data acquisition devices equipped with A/D (analog-to-digital) converters [1] allow computers to convert an electrical signal into digital form by frequently sampling the amplitude of the signal and storing the digitized values in memory. D/A (digital-to-analog) converters perform the reverse transformation: they generate electrical currents from the digitized amplitudes stored in the computer. These devices allow the computer to receive data from all kinds of instruments and to serve as a universal control device in industrial applications and scientific experiments.

Input and output devices are connected to the computer via hardware interface modules that implement specific data transfer protocols. In a personal computer, the interfaces may be built into the motherboard or take the form of special adapter cards that plug into special sockets on the motherboard, called extension slots. Devices connected to the computer are usually controlled by special programs called drivers that handle all the details and peculiarities of the device and the data transfer protocol. Today’s personal computers are equipped with several USB (Universal Serial Bus) ports, used for connecting virtually all peripheral devices: mice, printers, digital cameras, external disk drives, and so on. A high-speed USB interface can handle a data transmission rate of up to 480 megabits per second. (The emerging Superspeed USB 3.0 will increase the transmission rate tenfold.)
1.3 Software Overview

The term *software* refers to computer programs; it is also used as an adjective to refer to tasks or functions implemented through programs, as in “software interface,” “software fonts,” and so on. The line between hardware and software is not always clear. In the modern world, microprocessors are embedded in many objects, from microwave ovens and DVD players to cars and satellites. Their programs are developed using simulation tools on normal computers; when a program is finalized, it is permanently “burned” into ROMs. Such programs are referred to as *firmware*.

A modern computer not only runs individual programs but also maintains a “software environment.” This environment involves several functional layers (Figure 1-3). The bottom layer in this environment comprises BIOS, device drivers, interrupt handlers, etc. — programs that directly support hardware devices and functions. The next layer is the *operating system*, a software program that provides computer access services to users and standard support functions to other programs. The top layer is *software applications* (word processors, Internet browsers, business, industrial, or scientific applications, games, etc.).

![Software functional layers](image)

**Figure 1-3. Software functional layers**

The operating system loads programs into RAM from secondary storage and runs them. On a mainframe, the operating system allows multiple users to work on the computer at once through *time sharing*. In such a multi-user system, one user may be slowly editing a file or entering data on a terminal using only a small fraction of the available CPU time. At the same time another program may be doing “number crunching.” A multi-user operating system allocates “time slices” to each program and automatically switches between them. The operating system prioritizes the jobs
and swaps segments of programs in and out of memory as needed. A personal computer assumes one user, but contemporary users enjoy multi-tasking operating systems that let them keep several programs active concurrently (for example, a word processor, an e-mail program, and a music player).

The operating system also establishes and maintains a file system in secondary storage. Files are organized into a branching structure of directories and subdirectories (also called folders). The operating system provides commands and utility programs for navigating through the directory tree. Part of the operating system is a set of routines (sets of instructions, callable from other programs) that provide standard service functions to programs. These include functions for creating, reading, and writing files. The operating system shell provides a set of user commands, including commands for displaying, copying, deleting, and printing files, executing programs, and so on. Modern operating systems use GUI (Graphical User Interface, pronounced “gooey”), where commands can be entered by selecting items in menus or by clicking a mouse on an icon that represents a command or an object graphically.

Different types of operating systems include single-user, multi-user, multi-tasking, real-time, distributed, embedded, and so on. Descriptions and comparison of different types of operating systems can be found on the Internet [1, 2].

The top layer of software consists of application programs that make computers useful to people. In the old days, application programs mostly performed calculations or processed files. If interaction with the user was required, it was accomplished via an unsophisticated text dialog: a computer would display questions (or prompts) in plain text and let the user enter responses, then print out the resulting numbers, tables, or reports. Such applications are called console applications. In a modern computing environment, users expect programs to have GUI. The shift from console applications to GUI applications has brought changes in the way programs are designed and written and brought about new software development tools and methodologies, such as OOP (Object-Oriented Programming). Popular applications include productivity tools, web browsers and add-ons, games, media players, software development tools, and so on.

1.4 What do Software Engineers Do?

Don’t they write computer software? In brief, yes, of course. But this simple answer no longer explains what the computer programming (or, to sound more lofty, “software engineering”) profession has become. Software professionals create intricate and immensely complex invisible and intangible worlds. These worlds have
their laws, their architectures, their aesthetics. These worlds also have “windows” into the “real” world. So many different skills are involved that computer programming has become one of the most versatile and demanding professions. We are not talking about specific technical skills, listed by cryptic acronyms in a typical employment ad for a software engineer. They are important, of course, but they are changing so rapidly that many programmers just learn them on the job. What we are talking about are the core skills that define the profession.

The most important skill is, probably, the ability to absorb new technical information all the time and to put it to work. In no other technical field are things changing so fast. Some of the changes are driven by new hardware and the new capabilities it provides; others by user demands, and some simply by fashion.

A programmer also needs to be an architect. Computers manipulate data and follow specific instructions written by a programmer. To make a computer perform useful tasks, a programmer must represent both the data and the instructions in coherent structures and procedures. Consider, for example, a word processor application. Before you proceed to write code for it, you need to have an abstract model of a “document.” You consider: a document may have text, drawings, images, links, and so on. The user has to be able to edit a document, cut and paste pieces, and so on. Each of these elements and procedures must be reflected in the structure of your program. Sound software design makes it easier to implement and test the program, modify it when necessary, share the work among a team of programmers, and reuse pieces of the software in future projects.

Programmers are not building cities, bridges, or cathedrals, of course, and you won’t find pieces of computer software exhibited in a museum. Very few people in the world will be able to examine the software structures that a programmer creates. But if you could enter and walk around software systems, you would find all kinds of things, from rather elegant structures to huge monstrosities, often barely standing, with pieces held together by the software equivalent of string and duct tape. A small change in one place upsets the whole structure; more strings and tape and props become necessary to prevent the contraption from collapsing. Perhaps these specimens do belong in a museum after all! For better or worse, software remains mostly invisible, hidden behind thousands of lines of code. It is mysterious, sometimes even full of surprises (Figure 1-4).

Another important skill: a programmer needs to be able to understand algorithms and devise his own. An algorithm is a more or less abstract, formal, and general step-by-step recipe that tells how to perform a certain task or solve a certain problem on a computer. We will discuss algorithms in Chapter 4.
The next programmer’s skill is using programming languages. People often refer to programmers as “a C++ programmer,” “a Java programmer,” and so on. They think that knowing a particular programming language is what the programming profession is about. This is not so. First, a programmer is likely to encounter several programming languages over his or her career. Second, as we have seen, a programming job has many different aspects, and mastering a particular language is just one of them.

Still, ultimately a programmer needs to be able to express his or her brilliant structural designs and clever algorithms in a written program. A program written in a high-level language must obey the very precise syntax rules of that language and must also follow less precise but as important stylistic conventions established among professionals. We discuss Java syntax and style in Chapter 5.

But even when a program compiles with no errors, there is no guarantee that it works as desired. It may show unexpected behavior, produce incorrect output, or just crash. The programmer then has to figure out what is wrong and fix the problem. This takes yet another important skill: solving mysteries. A programmer must analyze the behavior of the program, hypothesize what may be wrong, either in the algorithm or in the code, then verify his hypothesis and make corrections. Of course, there are software tools, called debuggers, that help programmers locate bugs (errors) in their programs. But it would be too slow and tedious to use a debugger for every little error. Like a good doctor, a good programmer must develop an intuition for diagnosing and locating bugs quickly.
Believe it or not, there was time when programmers wrote their programs on paper and sent them to keypunch operators who would put them on punch cards [1]. Now programmers use sophisticated software development tools. Being able to use these tools proficiently is another skill in the programmer’s skill set.

In addition to general tools, a development environment for a particular programming language includes libraries of pre-programmed and pre-tested reusable software modules. There is no way to remember their technical details, of course. For example, in the 6.0 release, the standard Java library documentation consisted of 12,485 files in 844 folders and took 289 MB of disk space (which is roughly equivalent to 150,000 pages). Luckily the documentation is well organized and available online, so programmers can quickly find the information they need. But like a librarian, a programmer must know where and how to look.

Sometimes programmers forget that there are other people in the world who are not programmers, and that programs are written mostly for those other people, “users.” A programmer must make sure that his or her software is “user-friendly,” that is, easy to use and appears logical and intuitive to a non-technical user. Designing a friendly and effective user interface is another very important programming skill. A good user interface designer knows how to lay out screens, use colors, sound, animation, menus, keyboard commands, printouts and reports, and so on. The user interface is the only part of your software that is visible to the outside world, and this is often how your program is judged by non-technical users.

Last but not least, programmers must be familiar with the ethical codes for their profession and strive to uphold the highest professional ethical standards (see Appendix F).

To summarize, these are some of the things that a programmer must be able to do:

- Absorb and use emerging technical information
- Create sound software system architectures
- Understand and devise effective algorithms
- Be proficient with the syntax and style of programming languages
- Diagnose and correct programming errors
- Use software development tools and documentation
- Find and utilize reusable software components
- Design and implement friendly user interfaces
- Uphold the highest standards of professional ethics
Of course, when we said “A programmer must be able to do this; a programmer should know that,” we meant an abstract “programmer” who does not exist in the real world. In reality, a number of people are involved in every software development project, and different people may possess different skills. One person may specialize in designing system architectures, another person may be good at devising complex algorithms, a third person can quickly write and debug code, while someone else may not know much about programming but be an excellent designer of user interfaces. In the software development enterprise, there is a need for different types of work and room for people with different interests and skills.

The discipline that focuses on all aspects of software development is called computer science. A computer scientist takes a more theoretical view of one or another aspect of software development, looking for general ways to improve the efficiency of algorithms or for better software development methodologies and new programming languages. Unlike the computer scientist, a software engineer (or programmer) works on specific software development projects.

### 1.5 Representation of Information in Computer Memory

Computer memory is a uniform array of bytes that does not privilege any particular type of information. The memory can contain CPU instructions, numbers and text characters, and any other information that can be represented in digital form. Since a suitable analog/digital (A/D) converter can fairly accurately convert any electrical signal to digital form, any information that can be carried over a wire can be represented in computer memory. This includes sounds, images, motion, and so on (but, so far, excludes taste and smell).

CPU instructions are represented in computer memory in a manner specific to each particular brand of CPU. The first byte or two represent the operation code that identifies the instruction and the total number of bytes in that instruction; the following bytes may represent the values or memory addresses of the operands. How memory addresses are represented depends on the CPU architecture, but they are basically numbers that indicate the absolute sequential number of the byte in memory. A CPU may have special index registers that help calculate the actual address in memory for a specified instruction or operand.

The format for numbers is mostly dictated by the CPU, too, because the CPU has instructions for arithmetic operations that expect numbers to be represented in a certain way. Characters (letters, digits, etc.) are represented using one of the several character code systems that have become standard not only for representing text inside computers but also in computer terminals, printers, and other devices. The code assigns each character a number.
Fortunately, high-level programming languages, such as Java, shield computer programmers from the intricacies of how to represent CPU instructions, memory addresses, numbers, and characters.

Representing other types of information is often a matter of a specific application’s design. A black-and-white image, for example, may be represented as a sequence of bytes where each bit represents a pixel of the image: 0 for white and 1 for black. The sequence of pixels typically goes from left to right along each horizontal line of the image and then from top to bottom by row. Other memory locations may hold numbers that represent the image dimensions.

### 1.5.1 Numbers

Integers from 0 to 255 can be represented in one byte using the binary (base-2) system as follows:

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00000000</td>
</tr>
<tr>
<td>1</td>
<td>00000001</td>
</tr>
<tr>
<td>2</td>
<td>00000010</td>
</tr>
<tr>
<td>3</td>
<td>00000011</td>
</tr>
<tr>
<td>4</td>
<td>00000100</td>
</tr>
<tr>
<td>5</td>
<td>00000101</td>
</tr>
<tr>
<td>6</td>
<td>00000110</td>
</tr>
<tr>
<td>7</td>
<td>00000111</td>
</tr>
<tr>
<td>8</td>
<td>00001000</td>
</tr>
<tr>
<td>9</td>
<td>00001001</td>
</tr>
<tr>
<td>10</td>
<td>00001010</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>252</td>
<td>11111100</td>
</tr>
<tr>
<td>253</td>
<td>11111101</td>
</tr>
<tr>
<td>254</td>
<td>11111110</td>
</tr>
<tr>
<td>255</td>
<td>11111111</td>
</tr>
</tbody>
</table>
If we use 2 bytes (16 bits), we can represent integers from 0 to \(2^{16}-1 = 65535\):

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00000000 00000000</td>
</tr>
<tr>
<td>1</td>
<td>00000000 00000001</td>
</tr>
<tr>
<td>2</td>
<td>00000000 00000010</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>65534</td>
<td>11111111 11111110</td>
</tr>
<tr>
<td>65535</td>
<td>11111111 11111111</td>
</tr>
</tbody>
</table>

In general, \(k\) bits can produce \(2^k\) different combinations of 0s and 1s. Therefore, \(k\) bits used as binary digits can represent non-negative integers in the range from 0 to \(2^k-1\). A 32-bit memory address can identify \(2^{32} = 4,294,967,296\) different memory locations. So if we want to be able to address each individual byte, 32-bit addresses cover 4 GB of memory space.

CPU’s perform all their arithmetic operations on binary numbers. A CPU may have instructions that perform 8-bit, 16-bit, or 32-bit arithmetic, for instance. Since it is difficult for a human brain to grasp long sequences of 0s and 1s, programmers who have to deal with binary data often use the hexadecimal (or simply “hex”) representation in their documentation and programs. The hex system is the base-16 system, which uses 16 digits. The first ten digits are the usual 0 through 9, with the eleventh through sixteenth digits represented by the letters A through F. A byte can be split into two four-bit quads; each quad represents one hex digit, as follows:

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
<th>Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
<td>A</td>
</tr>
<tr>
<td>11</td>
<td>1011</td>
<td>B</td>
</tr>
<tr>
<td>12</td>
<td>1100</td>
<td>C</td>
</tr>
<tr>
<td>13</td>
<td>1101</td>
<td>D</td>
</tr>
<tr>
<td>14</td>
<td>1110</td>
<td>E</td>
</tr>
<tr>
<td>15</td>
<td>1111</td>
<td>F</td>
</tr>
</tbody>
</table>
Experienced programmers remember the bit patterns for the sixteen hex digits and can easily convert a binary number into hex and back. It is often convenient to use hex representation as an intermediate step for converting numbers from binary to decimal and back. For example:

\[
700_{10} = 2 \cdot 16^2 + 11 \cdot 16 + 12 = 2BC_{16} = 001010111100_2
\]

\[
1101101_2 = 6D_{16} = 6 \cdot 16 + 13 = 109_{10}
\]

The following examples show a few numbers represented in the decimal, hex, and 16-bit binary systems:

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hex</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>00000000 00000000</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>00000000 00000001</td>
</tr>
<tr>
<td>12</td>
<td>00C</td>
<td>00000000 00001100</td>
</tr>
<tr>
<td>32</td>
<td>020</td>
<td>00000000 00100000</td>
</tr>
<tr>
<td>128</td>
<td>080</td>
<td>00000000 10000000</td>
</tr>
<tr>
<td>255</td>
<td>0FF</td>
<td>00000000 11111111</td>
</tr>
<tr>
<td>256</td>
<td>100</td>
<td>00000000 00000000</td>
</tr>
<tr>
<td>32767</td>
<td>7FF</td>
<td>01111111 11111111</td>
</tr>
<tr>
<td>32768</td>
<td>8000</td>
<td>10000000 00000000</td>
</tr>
<tr>
<td>65535</td>
<td>FFFF</td>
<td>11111111 11111111</td>
</tr>
</tbody>
</table>

For a software developer, knowing the hex system is a matter of cultural literacy. In practice, programmers who use a high-level programming language, like Java, don’t have to use it very often and there are calculators and programs that can do conversions.

What about negative numbers? The same bit pattern may represent an unsigned (positive) integer and a negative integer, depending on how a particular instruction interprets it. Suppose we use 32-bit binary numbers, but now we decide that they represent signed integers. Positive integers from 0 to \(2^{31} - 1\) can be represented as before. These use only the 31 least significant bits. As to negative integers, their representation may be machine-dependent, varying from CPU to CPU. Many CPUs, including the Intel family, use a method called two’s-complement arithmetic. In this method, a negative integer \(x\) in the range from \(-1\) to \(-2^{31}\) is represented the same way as the unsigned binary number \(2^{32} - |x|\) where \(|x|\) is the absolute value of \(x\). For example, 21 and \(-21\) will be represented as:
Real numbers are represented using one of the standard formats expected by the CPU (or a separate floating-point arithmetic unit). Like scientific notation, this representation consists of a fractional part (mantissa) and an exponent part, but here both parts are represented as binary numbers. The IEEE (Institute of Electrical and Electronics Engineers) standard for a 4-byte (32-bit) representation uses 1 bit for the sign, 8 bits for the exponent and 23 bits for the mantissa. 127 is added to the exponent to ensure that negative exponents are still represented by non-negative numbers. This format lets programmers represent numbers in the range from approximately $-3.4 \times 10^{38}$ to $3.4 \times 10^{38}$ with at least seven digits of precision. Figure 1-5 gives a few examples.

![IEEE standard representation of 32-bit floating-point numbers](image)

**Figure 1-5.** IEEE standard representation of 32-bit floating-point numbers
There are dozens of applets on the Internet that illustrate binary representation of numbers and conversions from decimal to binary.

### 1.5.2 Characters

Characters are represented by numeric codes. These days most applications use platform-independent Unicode standard for encoding characters [1]. For example, the reason you can see the sentence written in Russian — Многие программы используют Юникод [1] — is that the software that displays this document understands the Unicode representation of Cyrillic characters.

**Java programs use Unicode internally for representing characters and text strings.**

Unicode uses two bytes per character. It can encode up to 65,000 characters, enough to encode the alphabets of most world languages and many special characters. Unicode has a provision to extend the character set even further, into millions of different codes.

In the old days, the two most common character codes were EBCDIC (Extended Binary Coded Decimal Interchange Code) [1], used in IBM mainframes, and ASCII (American Standard Code for Information Interchange, pronounced as'kee), used in personal computers, printers and other devices. Both of these use one byte per character. In the PC world, the term *ASCII file* refers to a text file (in which characters are represented in ASCII code), as opposed to a *binary file* that may contain numbers, images, or any other digitized information. Normally you won’t find EBCDIC-encoded data on a PC unless the file originated on a mainframe.

Unicode includes ASCII codes as a subset (called “C0 Controls and Basic Latin” [1]. ASCII code defines 128 characters with codes from 0 to 127 and uses only the seven least-significant bits of a byte. Codes from 33 to 127 represent “printable” characters: digits, upper- and lowercase letters, punctuation marks, and so on. 32 (hex 20) is a space.

The first 32 ASCII codes (0-31) are reserved for special control codes. For example, code 13 (hex 0D) is “carriage return” (CR), 10 (hex 0A) is “line feed” (LF), 12 (hex 0C) is “form feed” (FF) and 9 (hex 09) is “horizontal tab” (HT). How control codes are used may depend to some extent on the program or device that processes them. A standard ASCII table, including the more obscure control codes, is presented in Figure 1-6.
<table>
<thead>
<tr>
<th>Hex</th>
<th>0 _</th>
<th>1 _</th>
<th>2 _</th>
<th>3 _</th>
<th>4 _</th>
<th>5 _</th>
<th>6 _</th>
<th>7 _</th>
</tr>
</thead>
<tbody>
<tr>
<td>_0</td>
<td>NUL</td>
<td>DEL</td>
<td>(SPACE)</td>
<td>0</td>
<td>@</td>
<td>P</td>
<td>\</td>
<td>p</td>
</tr>
<tr>
<td>_1</td>
<td>SOH</td>
<td>DC1</td>
<td>!</td>
<td>1</td>
<td>A</td>
<td>Q</td>
<td>a</td>
<td>q</td>
</tr>
<tr>
<td>_2</td>
<td>STX</td>
<td>DC2</td>
<td>&quot;</td>
<td>2</td>
<td>B</td>
<td>R</td>
<td>b</td>
<td>r</td>
</tr>
<tr>
<td>_3</td>
<td>ETX</td>
<td>DC3</td>
<td>#</td>
<td>3</td>
<td>C</td>
<td>S</td>
<td>c</td>
<td>s</td>
</tr>
<tr>
<td>_4</td>
<td>EOT</td>
<td>DC4</td>
<td>$</td>
<td>4</td>
<td>D</td>
<td>T</td>
<td>d</td>
<td>t</td>
</tr>
<tr>
<td>_5</td>
<td>ENQ</td>
<td>NAK</td>
<td>%</td>
<td>5</td>
<td>E</td>
<td>U</td>
<td>e</td>
<td>u</td>
</tr>
<tr>
<td>_6</td>
<td>ACK</td>
<td>SYN</td>
<td>&amp;</td>
<td>6</td>
<td>F</td>
<td>V</td>
<td>f</td>
<td>v</td>
</tr>
<tr>
<td>_7</td>
<td>BEL</td>
<td>ETB</td>
<td>'</td>
<td>7</td>
<td>G</td>
<td>W</td>
<td>g</td>
<td>w</td>
</tr>
<tr>
<td>_8</td>
<td>BS</td>
<td>CAN</td>
<td>(</td>
<td>8</td>
<td>H</td>
<td>X</td>
<td>h</td>
<td>x</td>
</tr>
<tr>
<td>_9</td>
<td>HT</td>
<td>EM</td>
<td>)</td>
<td>9</td>
<td>I</td>
<td>Y</td>
<td>i</td>
<td>y</td>
</tr>
<tr>
<td>_A</td>
<td>LF</td>
<td>SUB</td>
<td>*</td>
<td>:</td>
<td>J</td>
<td>Z</td>
<td>j</td>
<td>z</td>
</tr>
<tr>
<td>_B</td>
<td>VT</td>
<td>ESC</td>
<td>+</td>
<td>;</td>
<td>K</td>
<td>[</td>
<td>k</td>
<td>{</td>
</tr>
<tr>
<td>_C</td>
<td>FF</td>
<td>FS</td>
<td>,</td>
<td>&lt;</td>
<td>L</td>
<td>\</td>
<td>l</td>
<td></td>
</tr>
<tr>
<td>_D</td>
<td>CR</td>
<td>GS</td>
<td>-</td>
<td>=</td>
<td>M</td>
<td>]</td>
<td>m</td>
<td>}</td>
</tr>
<tr>
<td>_E</td>
<td>SO</td>
<td>RS</td>
<td>.</td>
<td>&gt;</td>
<td>N</td>
<td>^</td>
<td>n</td>
<td>~</td>
</tr>
<tr>
<td>_F</td>
<td>SI</td>
<td>US</td>
<td>/</td>
<td>?</td>
<td>O</td>
<td>_</td>
<td>o</td>
<td>(NUL)</td>
</tr>
</tbody>
</table>

**Figure 1-6.** ASCII code used in personal computers and printers
1.8 Summary

Digital electronics represents information using two states: “on” or “off,” “high” or “low,” “1” or “0.” Digital devices, called gates, implement simple logical operations on signals: AND, OR, NOT. All other logical and arithmetic operations can be implemented using these three simple operations.

The heart of a computer is a CPU (central processing unit) that can perform logical and arithmetic operations. An executable program is a sequence of CPU instructions in machine code. It must be loaded into RAM (random-access memory) before it can run. All instructions and data addresses are encoded in binary sequences of 0s and 1s. The CPU fetches instructions and data from RAM, interprets operation codes, and executes the instructions.
RAM is arranged into bytes; each byte is 8 bits; each bit can hold either 1 or 0. The size of RAM is measured in kilobytes (1 KB = $2^{10}$ = 1024 bytes), megabytes (1 MB = $2^{20}$ = 1,048,576 bytes), or gigabytes (1 GB is over one billion bytes). The contents of RAM are erased when the power is turned off.

Mass storage devices have a larger memory capacity — hundreds of gigabytes or terabytes (1 TB is over one trillion bytes) — and they can hold the information permanently, but access to them is slower. Data in mass storage is arranged into files by the operating system software. The files are stored in a branching system of directories (represented as folders). The operating system also loads and runs applications, provides a GUI (graphical user interface) to users, and provides system services to programs (such as reading and writing files, supporting input devices, etc.).

All kinds of information are represented in the computer memory as sequences of 0s and 1s. Integers are represented as binary numbers. Real numbers use standard floating-point binary representation. Characters are represented as their codes in one of the standard coding systems. The most common codes are ASCII and Unicode. The latter includes thousands of characters from virtually all world alphabets.
Exercises

1. Name three types of operating systems and discuss the differences between them and where they are used.

2. Name several software applications from different categories that are installed on your computer.

3. Name three most powerful supercomputers in the world.

4. How does the computing power of a typical smartphone in 2014 compares to the computing power of a large mainframe computer of 1990s?

5. True or false?
   (a) A network router can be used to connect several networks of different types. ______
   (b) Network switches allow you to share resources on a LAN more efficiently. ______
   (c) Switches and routers are examples of hosts. ______
   (d) A server is usually implemented as a browser add-on. ______
   (e) “Workgroup” is a feature in Microsoft Windows operating systems, which allows to connect several computers into a network.

6. What does MAC stand for?

7. How many bytes are used in an IP address in the Version 4 standard? In the Version 6 standard?

8. Describe the differences between a single-user system, a client-server network and a peer-to-peer network.

Additional exercises for this chapter are in the Java Methods main text, Page 5.
Additional Exercises

1. Mark T (true) or F (false) the output of each of the following circuits with the given inputs.

\[\begin{array}{ccc}
\text{a) } & \text{T} & \text{F} \\
\text{b) } & \text{F} & \text{T} \\
\text{(c) } & \text{T} & \text{T}
\end{array}\]

2. Let’s say that two circuits are equivalent if they produce the same outputs for the same inputs. Draw a circuit equivalent to the one in Question 1-b using two NOT gates and one AND gate. ✔

3. Simplify the circuit in Question 1-c to an equivalent one that has only two gates: one NOT gate and one AND gate.
4. (a) Draw an alternative XOR circuit, different from the one in Figure 1-2, using two NOT gates, two OR gates, and one AND gate. \( \diamond \) Hint: at least one of the inputs, A or B, must be true, and at least one of the negated inputs, NOT A or NOT B, must be true, too. \( \diamond \checkmark \)

(b) Draw a third XOR circuit using four gates: one OR gate, two AND gates, and one NOT gate.

5. (MC) Computer memory is called RAM because:

A. It provides rapid access to data.
B. It is mounted on the motherboard.
C. It is measured in megabytes.
D. Its bytes can be addressed in random order.
E. Its chips are mounted in a rectangular array.

6. Mark true or false and explain:

(a) One meg of RAM can hold exactly as much information as one meg on a flash drive. _____

(b) A factory-formatted hard disk is split into a fixed number of files. _____ \( \checkmark \)

(c) In personal computers the operating system resides in ROM. _____ \( \checkmark \)

7. Find an old discarded desktop computer, **unplug the power cord**, and disconnect all other cables. Open the cover and identify the motherboard, CPU, RAM, USB ports (sockets for cable connectors), hard disk, CD-ROM, and other components and adapters, if present.

8. Identify the following entities or devices as part of a computer system’s hardware (H) or software (S).

(a) Operating system _____
(b) CPU _____
(c) GUI (Graphical User Interface) _____ \( \checkmark \)
(d) Bus _____
(e) RAM _____
(f) File _____
9. Identify the operating system that is running on your current computer and some software applications installed on it: a word processor, an Internet browser, a spreadsheet program, e-mail, an image processing application, games, and so on.

Section 1.5

10. Mark true or false:

(a) Only data but not CPU instructions can be stored in RAM. _____
(b) In ASCII code each character is represented in one byte. _____ √
(c) 16-bit binary numbers can be used to represent all non-negative integers from 0 to $2^{16} - 1$. ______
(d) Programs stored in ROM are referred to as “firmware.” _____

11. What is the maximum number of different codes or numbers that can be represented in

(a) 3 bits? ______ √
(b) 8 bits? ______
(c) 2 bytes? ______

12. Assuming that binary numbers represent unsigned integers in the usual way, with the least significant bit on the right, write the decimal value and the hex representation of the following binary numbers. Example:

<table>
<thead>
<tr>
<th>Binary</th>
<th>Decimal</th>
<th>Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>00001000</td>
<td>8</td>
<td>08</td>
</tr>
<tr>
<td>00011100</td>
<td>28</td>
<td>1C</td>
</tr>
</tbody>
</table>

(a) 00000010
(b) 00000111
(c) 10000000
(d) 00001011
(e) 11000011
(f) 11110101
(g) 00000101 10010010
13. An experiment consists of tossing a coin 10 times and its outcome is a sequence of heads and tails. How many possible outcomes are there?

14. How much memory does it take to hold a 512 by 512 gray-scale image with 256 levels of gray?

15. When a printer runs out of paper, the eight-bit printer status register of the parallel interface adapter gets the following settings: bit 7 (leftmost bit), "BUSY," is set to 1; bit 5, "PE" ("paper end"), is set to 1; and bit 3, "ERROR," is set to 0. Bit 4 is always 1 when a printer is connected; bit 6 is 0; and bits 0-2 are not used. Write the hex value equal to the setting of the printer status register when the printer runs out of paper, assuming that bits 0-2 are 0.

16. Design a method for representing the state of a tic-tac-toe board in computer memory. Can you fit your representation into three bytes?

17. In the game of Nim, stones are arranged in piles of arbitrary size. Each player in turn takes a few stones from any one pile. Every player must take at least one stone on every turn. The player who takes the last stone wins. Games of this type always have a winning strategy. This strategy can be established by tagging all possible positions in the game with two tags, “plus” and “minus,” in such a way that any move from a “plus” position always leads to a “minus” position, and from any “minus” position there is always a possible move into some “plus” position. The final winning position must be tagged “plus.” Therefore, if the first player begins in a “minus” position, she can win by moving right away into a “plus” position and returning to a “plus” position on each subsequent move. If, however, the first player begins in a “plus” position, then the second player can win, provided he knows how to play correctly.

In Nim, we can convert the number of stones in each pile into a binary number and write these binary numbers in one column (so that the “units” digits are aligned on the right). We can tag the position “plus” if the number of 1s in each column is even and “minus” if the count of 1s in at least one column is odd. Prove that this method of tagging “plus” and “minus” positions defines a winning strategy. Who wins starting with four piles of 1, 3, 5, and 7 stones — the first or the second player? What’s the correct response if the first player takes five stones from the pile of 7?
The table below is called a *Greco-Roman square*: each of the three Latin letters occurs exactly once in each row and each column; the same is true for each of the three Greek letters; and each Latin-Greek combination occurs exactly once in the table:

\[
\begin{array}{ccc}
A\gamma & B\alpha & C\beta \\
B\beta & C\gamma & A\alpha \\
C\alpha & A\beta & B\gamma
\end{array}
\]

Substitute the digits 0, 1, and 2 for A, B, C and for \(\alpha, \beta, \gamma\) (in any order). Convert the resulting base-3 numbers into decimal (base-10) numbers. The base-3 system uses only three digits: 0, 1, and 2. The numbers are represented as follows:

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Base 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td>6</td>
<td>20</td>
</tr>
<tr>
<td>7</td>
<td>21</td>
</tr>
<tr>
<td>8</td>
<td>22</td>
</tr>
<tr>
<td>9</td>
<td>100</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Add 1 to each number. You will get a table in which the numbers 1 through 9 are arranged in such a way that the sum of the numbers in each row and column is the same. Explain why you get this result and find a way to substitute the digits 0, 1, and 2 for letters so that the sum of numbers in each of the two diagonals is also the same as in the rows and columns. What you get then is called a *magic square*. Using a similar method, build a 5 by 5 magic square.