2 BINARY VALUES AND NUMBER SYSTEMS

Now that we've established history and some common terminology in Chapter 1, our exploration of computing technology can begin in earnest. This chapter describes binary values—the way in which computer hardware represents and manages information. This chapter also puts binary values in the context of all number systems, reminding us of grade school concepts that we now take for granted. You probably already know many of the concepts about binary numbers described in this chapter, but you might not realize that you know them! The rules of all number systems are the same; it's just a matter of going back to those underlying concepts and applying them in a new base. By making sure we have an understanding of binary values, we pave the way to understanding how computing systems use the binary number system to accomplish their tasks.

GOALS

After studying this chapter, you should be able to:

- distinguish among categories of numbers.
- describe positional notation.
- convert numbers in other bases to base 10.
- convert base-10 numbers to numbers in other bases.
- describe the relationship between bases 2, 8, and 16.
- explain the importance to computing of bases that are powers of 2.

2.1 Numbers and Computing

Numbers are crucial to computing. In addition to using a computer to execute numeric computations, all types of information that we store and manage using a computer are ultimately stored as numbers. At the lowest level, computers store all information using just the digits 0 and 1. So to begin our exploration of computers, we need to first begin by exploring numbers.

First, let's recall that numbers can be classified into all sorts of categories. There are natural numbers, negative numbers, rational numbers, irrational numbers, and many others that are important in mathematics but not to the understanding of computing. Let's review the relevant category definitions briefly.

First, let's define the general concept of a number: A number is a unit belonging to an

abstract mathematical system and is subject to specified laws of succession, addition, and multiplication. That is, a number is a representation of a value, and certain arithmetic operations can be consistently applied to such values.

Number A unit of an abstract mathematical system subject to the laws of arithmetic

Now let's separate numbers into categories. A natural number is the number 0 or any number obtained by repeatedly adding 1 to this number. Natural numbers are the ones we use in counting. A negative number is less than zero and is opposite in sign to a positive number. An integer is any of the natural numbers or any of the negatives of these numbers. A rational number is an integer or the quotient of two integers—that is, any value that can be expressed as a fraction.

Natural number The number 0 and any number obtained by repeatedly adding 1 to it
Negative number A value less than 0, with a sign opposite to its positive counterpart
Integer A natural number, a negative of a natural number, or zero
Rational number An integer or the quotient of two integers (division by zero excluded)

In this chapter, we focus on natural numbers and the ways that they are represented in various number systems. As part of our discussion, we establish how all number systems relate to each other. In Chapter 3, we examine the computer representation of negative and rational numbers, as well as how we use numbers to represent other forms of data such as characters and images.

Some of the material in this chapter may already be familiar to you. Certainly some of the underlying ideas should be. You probably take for granted some basic principles of numbers and arithmetic because you've become so used to them. Part of our goal in this chapter is to remind you of those underlying principles and to show you that they apply to all number systems. Then the idea that a computer uses binary values—that is, 1s and 0s—to represent information should be less mysterious.

2.2 Positional Notation

How many ones are there in 943? That is, how many actual things does the number 943 represent? Well, in grade school terms, you might say there are 9 hundreds plus 4 tens plus 3 ones. Or, said another way, there are 900 ones plus 40 ones plus 3 ones. So how many ones are there in 754? 700 ones plus 50 ones plus 4 ones. Right? Well, maybe. The answer depends on the *base* of the number system you are using. This answer is correct in the base-10, or decimal, number system, which is the number system humans use every day. But that answer is not correct in other number systems.

The base of a number system specifies the number of digits used in the system. The digits always begin with 0 and continue through one less than the base. For example, there are 2 digits in base 2: 0 and 1. There are 8 digits in base 8: 0 through 7. There are 10 digits in base 10: 0 through 9. The base also determines what the positions of digits mean. When

you add 1 to the last digit in the number system, you have a carry to the digit position to the left.

> Base The foundational value of a number system, which dictates the number of digits and the value of digit positions

Numbers are written using positional notation. The rightmost digit represents its value multiplied by the base to the zeroth power. The digit to the left of that one represents its value multiplied by the base to the first power. The next digit represents its value multiplied by the base to the second power. The next digit represents its value multiplied by the base to the third power, and so on. You are so familiar with positional notation that you probably don't think about it. We used it instinctively to calculate the number of ones in 943.

> Positional notation A system of expressing numbers in which the digits are arranged in succession, the position of each digit has a place value, and the number is equal to the sum of the products of each digit by its place value[1]

$$\begin{aligned} 9 * 10^2 = 9 * 100 &= 900 \\ + 4 * 10^1 = 4 * \ 10 &= \ 40 \\ + 3 * 10^0 = 3 * \ \ 1 &= \ \ 3 \\ &\overline{\quad 943} \end{aligned}$$

> ?
>
> The importance of zero
>
> Positional notation is possible only because of the concept of zero. Zero was the fundamental concept at the intersection of all branches of modern mathematics. As Georges Ifrah noted in his book *The Universal History of Computing:* "To sum up, the vital discovery of zero gave the human mind an extraordinarily powerful potential. No other human creation has exercised such an influence on the development of mankind's intelligence."[2]

A more formal way of defining positional notation is to say that the value is represented as a polynomial in the base of the number system. But what is a polynomial? A polynomial is a sum of two or more algebraic terms, each of which consists of a constant multiplied by one or more variables raised to a nonnegative integral power. When defining positional notation, the variable is the base of the number system. Thus 943 is represented as a polynomial as follows, with x acting as the base:

$$9 * x^2 + 4 * x^1 + 3 * x^0$$

Let's express this idea formally. If a number in the base-R number system has n digits, it is represented as follows, where d_i represents the digit in the ith position in the number:

$$d_n * R^{n-1} + d_{n-1} * R^{n-2} + \cdots + d_2 * R + d_1$$

Look complicated? Let's look at a concrete example: 63578 in base 10. Here n is 5 (the

number has five digits), and *R* is 10 (the base). The formula says that the fifth digit (last digit on the left) is multiplied by the base to the fourth power; the fourth digit is multiplied by the base to the third power; the third digit is multiplied by the base to the second power; the second digit is multiplied by the base to the first power; and the first digit is not multiplied by anything.

$$6 * 10^4 + 3 * 10^3 + 5 * 10^2 + 7 * 10^1 + 8$$

In the previous calculation, we assumed that the number base is 10. This is a logical assumption because our number system *is* base 10. However, there is no reason why the number 943 couldn't represent a value in base 13. If so, to determine the number of ones, we would have to convert it to base 10.

$$\begin{aligned} 9 * 13^2 &= 9 * 169 = 1521 \\ +\,4 * 13^1 &= 4 * 13 = 52 \\ +\,3 * 13^0 &= 3 * 1 = 3 \\ &\qquad\qquad\quad 1576 \end{aligned}$$

Therefore, 943 in base 13 is equal to 1576 in base 10. Keep in mind that these two numbers have an equivalent value. That is, both represent the same number of things. If one bag contains 943 (base 13) beans and a second bag contains 1576 (base 10) beans, then both bags contain the exact same number of beans. Number systems just allow us to represent values in various ways.

Note that in base 10, the rightmost digit is the "ones" position. In base 13, the rightmost digit is also the "ones" position. In fact, this is true for any base, because anything raised to the power 0 is 1.

Why would anyone want to represent values in base 13? It isn't done very often, granted, but it is sometimes helpful to understand how it works. For example, a computing technique called *hashing* takes numbers and scrambles them, and one way to scramble numbers is to interpret them in a different base.

Other bases, such as base 2 (binary), are particularly important in computer processing. Let's explore these bases in more detail.

Binary, Octal, and Hexadecimal

The base-2 (binary) number system is particularly important in computing. It is also helpful to be familiar with number systems that are powers of 2, such as base 8 (octal) and base 16 (hexadecimal). Recall that the base value specifies the number of digits in the number system. Base 10 has ten digits (0–9), base 2 has two digits (0–1), and base 8 has eight digits (0–7). Therefore, the number 943 could not represent a value in any base less than base 10, because the digit 9 doesn't exist in those bases. It is, however, a valid number in base 10 or any base higher than that. Likewise, the number 2074 is a valid number in base 8 or higher, but it simply does not exist (because it uses the digit 7) in any base lower than that.

What are the digits in bases higher than 10? We need symbols to represent the digits that correspond to the decimal values 10 and beyond. In bases higher than 10, we use letters as digits. We use the letter A to represent the number 10, B to represent 11, C to

represent 12, and so forth. Therefore, the 16 digits in base 16 are:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Let's look at values in octal, hexadecimal, and binary to see what they represent in base 10. For example, let's calculate the decimal equivalent of 754 in octal (base 8). As before, we just expand the number in its polynomial form and add up the numbers.

$$\begin{aligned} 7 * 8^2 &= 7 * 64 = 448 \\ +\,5 * 8^1 &= 5 * 8 = 40 \\ +\,4 * 8^0 &= 4 * 1 = 4 \\ & \qquad\quad 492 \end{aligned}$$

Let's convert the hexadecimal number ABC to decimal:

$$\begin{aligned} A * 16^2 &= 10 * 256 = 2560 \\ +\,B * 16^1 &= 11 * 16 = 176 \\ +\,C * 16^0 &= 12 * 1 = 12 \\ & \qquad\quad 2748 \end{aligned}$$

Note that we perform the exact same calculation to convert the number to base 10. We just use a base value of 16 this time, and we have to remember what the letter digits represent. After a little practice you won't find the use of letters as digits that strange.

Finally, let's convert a binary (base-2) number 1010110 to decimal. Once again, we perform the same steps—only the base value changes:

$$\begin{aligned} 1 * 2^6 &= 1 * 64 = 64 \\ +\,0 * 2^5 &= 0 * 32 = 0 \\ +\,1 * 2^4 &= 1 * 16 = 16 \\ +\,0 * 2^3 &= 0 * 8 = 0 \\ +\,1 * 2^2 &= 1 * 4 = 4 \\ +\,1 * 2^1 &= 1 * 2 = 2 \\ +\,0 * 2^0 &= 0 * 1 = 0 \\ & \qquad\quad 86 \end{aligned}$$

The Abacus

In our brief history of computing in Chapter 1, we mentioned the abacus as an early computing device. More specifically, the abacus is a device that uses positional notation to represent a decimal number. The beads in any one column represent the digit in that column. All columns combined represent a complete number.

Courtesy of Theresa DiDonato

The beads above the middle bar represent units of 5 and the beads below the bar each represent 1. Beads pushed away from the middle bar do not contribute to the number. The following diagram shows the number 27,091 represented on an abacus:

Courtesy of Theresa DiDonato

The user performs calculations by moving the beads in specific ways to reflect the basic arithmetic operations of addition, subtraction, multiplication, and division.

Though ancient, the abacus is still used today in many Asian cultures. In stores, a checkout clerk might use an abacus instead of an electronic cash register. Although lacking some of the advantages of electronic devices, the abacus is more than sufficient for the kinds of calculations needed for basic business transactions. Skilled users of an abacus can rival anyone with a calculator in terms of both speed and accuracy.

Children in these cultures learn rote operations on the abacus, much as you were drilled in your multiplication tables. To perform an operation on a number, the user executes a series of movements using only the thumb, pointing finger, and middle finger of one hand. These movements correspond to individual digits and depend on the operation being performed. For example, to add the digit 7 to the digit 5 already showing on the abacus, the user clears the five marker (pushes it to the top), pushes 2 onto the bar from below, and increments 1 in the next column. Though this move corresponds to the basic addition operation we do on paper, the abacus user is not thinking about the mathematics. The user is conditioned to execute a specific movement when specific digits are encountered for a specific operation. When the calculation is complete, the user reads the result as shown on the abacus.

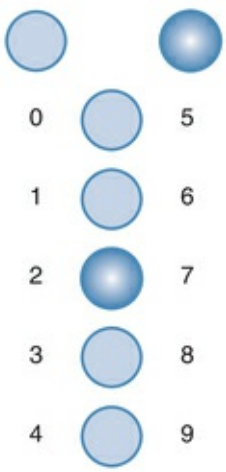
Recall that the digits in any number system go up to one less than the base value. To represent the base value in any base, you need two digits. A 0 in the rightmost position and a 1 in the second position represent the value of the base itself. Thus 10 is ten in base 10, 10 is eight in base 8, and 10 is sixteen in base 16. Think about it. The consistency of number systems is actually quite elegant.

Bi-quinary Number Representation

The console of the IBM 650, a popular commercial computer in the late 1950s, allowed the operator to read the contents of memory using the bi-quinary system. This number representation system uses seven lights to represent the 10 decimal digits.

Courtesy of IBM Corporate Archives, © International Business Machines Corporation

Each digit is represented by two lights, one of the top two and one of the bottom five. If the upper-left light is on, the five other lights represent 0, 1, 2, 3, and 4, respectively, from top to bottom. If the upper-right light is on, the five other lights represent 5, 6, 7, 8, and 9. The following configuration represents the number 7:



The IBM 650 was called the Ford Tri-Motor of computers: Like the Ford Tri-Motor, old IBM 650s were shipped to Latin America where they enjoyed an extended life.

Addition and subtraction of numbers in other bases are performed exactly like they are on decimal numbers.

Arithmetic in Other Bases

Recall the basic idea of arithmetic in decimal: 0 + 1 is 1, 1 + 1 is 2, 2 + 1 is 3, and so on. Things get interesting when you try to add two numbers whose sum is equal to or larger than the base value—for example, 1 + 9. Because there isn't a symbol for 10, we reuse the same digits and rely on position. The rightmost digit reverts to 0, and there is a carry into the next position to the left. Thus 1 + 9 equals 10 in base 10.

The rules of binary arithmetic are analogous, but we run out of digits much sooner. That is, 0 + 1 is 1, and 1 + 1 is 0 with a carry. Then the same rule is applied to every column in a larger number, and the process continues until we have no more digits to add. The example below adds the binary values 101110 and 11011. The carry value is marked

above each column in color.

```
  11111      ← carry
   101110
+   11011
  1001001
```

We can convince ourselves that this answer is correct by converting both operands to base 10, adding them, and comparing the result: 101110 is 46, 11011 is 27, and the sum is 73. Of course, 1001001 is 73 in base 10.

The subtraction facts that you learned in grade school were that 9 – 1 is 8, 8 – 1 is 7, and so on, until you try to subtract a larger digit from a smaller one, such as 0 – 1. To accomplish this feat, you have to "borrow one" from the next left digit of the number from which you are subtracting. More precisely, you borrow one power of the base. So, in base 10, when you borrow, you borrow 10. The same logic applies to binary subtraction. Every time you borrow in a binary subtraction, you borrow 2. Here are two examples with the borrowed values marked above.

```
    1
   0⁄2   ← borrow
  111001
-   110
  110011
```

```
   02
   ⁄2    ← borrow
  111101
-   110
  110111
```

Once again, you can check the calculation by converting all values to base 10 and subtracting to see if the answers correspond.

Power-of-2 Number Systems

Binary and octal numbers share a very special relationship: Given a number in binary, you can read it off in octal; given a number in octal, you can read it off in binary. For example, take the octal number 754. If you replace each digit with the binary representation of that digit, you have 754 in binary. That is, 7 in octal is 111 in binary, 5 in octal is 101 in binary, and 4 in octal is 100 in binary, so 754 in octal is 111101100 in binary.

To facilitate this type of conversion, the table below shows counting in binary from 0 through 10 with their octal and decimal equivalents.

BINARY	OCTAL	DECIMAL
0	0	0
1	1	1
10	2	2
11	3	3
100	4	4
101	5	5
110	6	6
111	7	7
1000	10	8
1001	11	9

?

Can you count to three?

Not instinctively! Cognitive psychologists have demonstrated that preschool children do not identify more than three sets: a set of one object, two objects, and three or more objects (also called *many*). Until some two centuries ago, numerous languages had only two or three number words: words for *single, pair*, and *many*. We still have such words in English: *gang, pile, bunch, flock, herd, school, fleet, pride, pack*, and *gaggle*.[3]

To convert from binary to octal, you start at the rightmost binary digit and mark the digits in groups of threes. Then you convert each group of three to its octal value.

111	101	100
7	5	4

Let's convert the binary number 1010110 to octal, and then convert that octal value to decimal. The answer should be the equivalent of 1010110 in decimal, or 86.

1	010	110
1	2	6

$$\begin{aligned} & 1 * 8^2 = 1 * 64 = 64 \\ + \; & 2 * 8^1 = 2 * 8 = 16 \\ + \; & 6 * 8^0 = 6 * 1 = 6 \\ & \qquad\qquad\quad 86 \end{aligned}$$

The reason that binary can be immediately converted to octal and octal to binary is that 8 is a power of 2. There is a similar relationship between binary and hexadecimal. Every hexadecimal digit can be represented in four binary digits. Let's take the binary number 1010110 and convert it to hexadecimal by marking the digits from right to left in groups of four.

101	0110
5	6

$$\begin{aligned} & 5 * 16^1 = 5 * 16 = 80 \\ + \; & 6 * 16^0 = 6 * 1 = 6 \\ & \qquad\qquad\quad 86 \end{aligned}$$

Now let's convert ABC in hexadecimal to binary. It takes four binary digits to represent each hex digit. A in hexadecimal is 10 in decimal and therefore is 1010 in binary. Likewise, B in hexadecimal is 1011 in binary, and C in hexadecimal is 1100 in binary. Therefore, ABC in hexadecimal is 101010111100 in binary.

Rather than confirming that 101010111100 is 2748 in decimal directly, let's mark it off in octal and convert the octal.

```
101   010   111   100
 5     2     7     4
```

Thus 5274 in octal is 2748 in decimal.

In the next section, we show how to convert base-10 numbers to the equivalent number in another base.

Converting from Base 10 to Other Bases

The rules for converting base-10 numbers involve dividing by the base into which you are converting the number. From this division, you get a quotient and a remainder. The remainder becomes the next digit in the new number (going from right to left), and the quotient replaces the number to be converted. The process continues until the quotient is zero. Let's write the rules in a different form.

```
WHILE (the quotient is not zero)
   Divide the decimal number by the new base
   Make the remainder the next digit to the left in the answer
   Replace the decimal number with the quotient
```

These rules form an *algorithm* for converting from base 10 to another base. An algorithm is a logical sequence of steps that solves a problem. We have much more to say about algorithms in later chapters. Here we show one way of describing an algorithm and then apply it to perform the conversions.

The first line of the algorithm tells us to repeat the next three lines until the quotient from our division becomes zero. Let's convert the decimal number 2748 to hexadecimal. As we've seen in previous examples, the answer should be ABC.

```
   171      ← quotient
16)2748
   16
   114
   112
     28
     16
     12     ← remainder
```

The remainder (12) is the first digit in the hexadecimal answer, represented by the digit C. So the answer so far is C. Since the quotient is not zero, we divide it (171) by the new base.

```
    10      ← quotient
16)171
   16
    11      ← remainder
```

The remainder (11) is the next digit to the left in the answer, which is represented by

the digit B. Now the answer so far is BC. Since the quotient is not zero, we divide it (10) by the new base.

```
    0        ← quotient
16)10
   0
   10        ← remainder
```

The remainder (10) is the next digit to the left in the answer, which is represented by the digit A. Now the answer is ABC. The quotient is zero, so we are finished, and the final answer is ABC.

Binary Values and Computers

Although some of the early computers were decimal machines, modern computers are binary machines. That is, numbers within the computer are represented in binary form. In fact, all information is somehow represented using binary values. The reason is that each storage location within a computer contains either a low-voltage signal or a high-voltage signal. Because each location can have only one of two states, it is logical to equate those states to 0 and 1. A low-voltage signal is equated with a 0, and a high-voltage signal is equated with a 1. In fact, you can forget about voltages and think of each storage location as containing either a 0 or a 1. Note that a storage location cannot be empty: It must contain either a 0 or a 1.

Grace Murray Hopper

From 1943 until her death on New Year's Day in 1992, Admiral Grace Murray Hopper was intimately involved with computing. In 1991, she was awarded the National Medal of Technology "for her pioneering accomplishments in the development of computer programming languages that simplified computer technology and opened the door to a significantly larger universe of users."

Admiral Hopper was born Grace Brewster Murray in New York City on December 9, 1906. She attended Vassar and received a PhD in mathematics from Yale. For the

next 10 years, she taught mathematics at Vassar.

In 1943, Admiral Hopper joined the U.S. Navy and was assigned to the Bureau of Ordnance Computation Project at Harvard University as a programmer on the Mark I. After the war, she remained at Harvard as a faculty member and continued work on the Navy's Mark II and Mark III computers. She loved to tell the story of how, while she was working on the Mark II, one of the operators discovered the first computer "bug"—a moth caught in one of the relays. In 1949, she joined Eckert-Mauchly Computer Corporation and worked on the UNIVAC I.

Admiral Hopper had a working compiler in 1952, a time when the conventional wisdom was that computers could do only arithmetic. Although not on the committee that designed the computer language COBOL, she was active in its design, implementation, and use. COBOL (which stands for *Common Business-Oriented Language*) was developed in the early 1960s and is still widely used in business data processing.

Admiral Hopper retired from the Navy in 1966, only to be recalled within a year to full-time active duty. Her mission was to oversee the Navy's efforts to maintain uniformity in programming languages. It has been said that just as Admiral Hyman Rickover was the father of the nuclear navy, Rear Admiral Hopper was the mother of computerized data automation in the Navy. She served with the Naval Data Automation Command until she retired again in 1986 with the rank of Rear Admiral. At the time of her death, she was a senior consultant at Digital Equipment Corporation.

Admiral Hopper loved young people and enjoyed giving talks on college and university campuses. She often handed out colored wires, which she called *nanoseconds* because they were cut to a length of about one foot—the distance that light travels in a nanosecond (billionth of a second). Her advice to the young was, "You manage things, you lead people. We went overboard on management and forgot about the leadership."

During her lifetime, Admiral Hopper received honorary degrees from more than 40 colleges and universities. She was honored by her peers on several occasions, including the first Computer Sciences Man of the Year award given by the Data Processing Management Association, and the Contributors to Computer Science Education Award given by the Special Interest Group for Computer Science Education (SIGCSE), which is part of the ACM (Association for Computing Machinery).

Nell Dale, when notifying Admiral Hopper of the SIGCSE award, asked of which of her many accomplishments she was most proud. She answered, "All the young people I have trained over the years."

Each storage unit is called a binary digit, or bit for short. Bits are grouped together into bytes (8 bits), and bytes are grouped together into units called words. The number of bits in a word is known as the word length of the computer. For example, IBM 370 architecture in the late 1970s had half words (2 bytes or 16 bits), full words (4 bytes), and double words (8 bytes).

Binary digit A digit in the binary number system; a 0 or a 1

Bit Binary digit

Byte Eight binary digits

Word A group of one or more bytes; the number of bits in a word is the word length of the computer

Modern computers are often 32-bit machines (such as Intel's Pentium IV processor) or 64-bit machines (such as Hewlett-Packard's Alpha processors and Intel's Itanium 2 processor). However, some microprocessors that are used in applications such as pagers are 8-bit machines. The computing machine you are using—whatever it is—is ultimately supported by the binary number system.

We have much more to explore about the relationship between computers and binary numbers. In the next chapter, we examine many kinds of data and see how they are represented in a computer. In Chapter 4, we see how to control electrical signals that represent binary values. In Chapter 6, we see how binary numbers are used to represent program commands that the computer executes.

SUMMARY

Numbers are written using positional notation, in which the digits are arranged in succession, the position of each digit has a place value, and the number is equal to the sum of the products of each digit by its place value. The place values are powers of the base of the number system. Thus, in the decimal number system, the place values are powers of 10; in the binary number system, the place values are powers of 2.

Arithmetic can be performed on numbers in any base represented in positional notation. The same operational rules apply to other bases as they do to base 10. Adding 1 to the largest digit in the base causes a carry into the next position.

Base 2, base 8, and base 16 are all related because these bases are powers of 2. This relationship provides a quick way to convert between numbers in these bases. Computer hardware is designed using numbers in base 2. A low-voltage signal is equated with 0, and a high-voltage signal is equated with 1.

ETHICAL ISSUES

The FISA Court

The United States Foreign Intelligence Surveillance Court is a U.S. federal court that was established under the Foreign Intelligence Surveillance Act of 1978 (FISA). The Court handles requests by federal law enforcement agencies for surveillance warrants against suspected foreign intelligence agents operating inside the United States.[4]

Before 2013, when Edward Snowden leaked that the Court had ordered a subsidiary of Verizon to provide detailed call records to the National Security Agency (NSA), most people had never heard of the FISA Court. The next chapter examines the controversy surrounding it.

The FISA Court comprises 11 judges who sit for 7-year terms. The Chief Justice of the Supreme Court appoints the judges, without confirmation. An application for an

electronic surveillance warrant is made before one of the judges. The court may amend this application before granting the warrant. If the application is denied, the government may not take the same request to another judge. If the U.S. Attorney General determines that an emergency exists, he or she may authorize the electronic surveillance but must notify a Court judge not more than 72 hours after the authorization. The USA PATRIOT Act of 2001 expanded the time periods during which surveillance may be authorized.[5]

In December 2012, President Obama signed the FISA Amendments Act Reauthorization Act of 2012, which extends Title VII of FISA until December 31, 2017.

Title VII of FISA, added by the FISA Amendments Act of 2008, created separate procedures for targeting suspected foreign intelligence agents, including non-U.S. persons and U.S. persons reasonably believed to be outside the United States.[6]

Note that the stated intent of the FISA Court is to protect the United States as well as the rights of U.S. citizens.

KEY TERMS

Base
Binary digit
Bit
Byte
Integer
Natural number
Negative number
Number
Positional notation
Rational number
Word

EXERCISES

For Exercises 1–5, match the following numbers with their definition.

A. Number
B. Natural number
C. Integer number
D. Negative number
E. Rational number

1. A unit of an abstract mathematical system subject to the laws of arithmetic
2. A natural number, a negative of a natural number, or zero
3. The number zero and any number obtained by repeatedly adding one to it
4. An integer or the quotient of two integers (division by zero excluded)

5. A value less than zero, with a sign opposite to its positive counterpart

For Exercises 6–11, match the solution with the problem.

A. 10001100
B. 10011110
C. 1101010
D. 1100000
E. 1010001
F. 1111000

6. 1110011 + 11001 (binary addition)
7. 1010101 + 10101 (binary addition)
8. 1111111 + 11111 (binary addition)
9. 1111111 – 111 (binary subtraction)
10. 1100111 – 111 (binary subtraction)
11. 1010110 – 101 (binary subtraction)

For Exercises 12–17, mark the answers true or false as follows:

A. True
B. False

12. Binary numbers are important in computing because a binary number can be converted into every other base.
13. Binary numbers can be read off in hexadecimal but not in octal.
14. Starting from left to right, every grouping of four binary digits can be read as one hexadecimal digit.
15. A byte is made up of six binary digits.
16. Two hexadecimal digits cannot be stored in one byte.
17. Reading octal digits off as binary produces the same result whether read from right to left or from left to right.

Exercises 18–47 are problems or short-answer questions.

18. Distinguish between a natural number and a negative number.
19. Distinguish between a natural number and a rational number.
20. Label the following numbers as natural, negative, or rational.
    a. 1.333333
    b. –1/3
    c. 1066
    d. 2/5
    e. 6.2
    f. π (pi)
21. How many ones are there in 891 if it is a number in each of the following bases?
    a. Base 10
    b. Base 8
    c. Base 12
    d. Base 13
    e. Base 16
22. Express 891 as a polynomial in each of the bases in Exercise 21.
23. Convert the following numbers from the base shown to base 10.

a. 111 (base 2)
b. 777 (base 8)
c. FEC (base 16)
d. 777 (base 16)
e. 111 (base 8)

24. Explain how base 2 and base 8 are related.
25. Explain how base 8 and base 16 are related.
26. Expand the table on page 43 to include the decimals from 11 through 16.
27. Expand the table in Exercise 26 to include hexadecimal numbers.
28. Convert the following binary numbers to octal.
    a. 111110110
    b. 1000001
    c. 10000010
    d. 1100010
29. Convert the following binary numbers to hexadecimal.
    a. 10101001
    b. 11100111
    c. 01101110
    d. 01121111
30. Convert the following hexadecimal numbers to octal.
    a. A9
    b. E7
    c. 6E
31. Convert the following octal numbers to hexadecimal.
    a. 777
    b. 605
    c. 443
    d. 521
    e. 1
32. Convert the following decimal numbers to octal.
    a. 901
    b. 321
    c. 1492
    d. 1066
    e. 2001
33. Convert the following decimal numbers to binary.
    a. 45
    b. 69
    c. 1066
    d. 99
    e. 1
34. Convert the following decimal numbers to hexadecimal.
    a. 1066
    b. 1939
    c. 1
    d. 998

e. 43

35. If you were going to represent numbers in base 18, which symbols might you use to represent the decimal numbers 10 through 17 other than letters?
36. Convert the following decimal numbers to base 18 using the symbols you suggested in Exercise 35.
    a. 1066
    b. 99099
    c. 1
37. Perform the following octal additions.
    a. 770 + 665
    b. 101 + 707
    c. 202 + 667
38. Perform the following hexadecimal additions.
    a. 19AB6 + 43
    b. AE9 + F
    c. 1066 + ABCD
39. Perform the following octal subtractions.
    a. 1066 – 776
    b. 1234 – 765
    c. 7766 – 5544
40. Perform the following hexadecimal subtractions.
    a. ABC – 111
    b. 9988 – AB
    c. A9F8 – 1492
41. Why are binary numbers important in computing?
42. How many bits does a byte contain?
43. How many bytes are there in a 64-bit machine?
44. Why do microprocessors such as pagers have only 8-bit words?
45. Why is it important to study how to manipulate fixed-size numbers?
46. How many ones are there in the number AB98 in base 13?
47. Describe how a bi-quinary number representation works.

THOUGHT QUESTIONS

1. Exercise 20 asked you to classify π as one of the options. π does not belong in any of the categories named; π (and e) are transcendental numbers. Look up *transcendental numbers* in the dictionary or in an old math book and give the definition in your own words.
2. Complex numbers are another category of numbers that are not discussed in this chapter. Look up *complex numbers* in a dictionary or an old math book and give the definition in your own words.
3. Many everyday occurrences can be represented as a binary bit. For example, a door is open or closed, the stove is on or off, and the dog is asleep or awake. Could relationships be represented as a binary value? Discuss the question, giving

examples.

4. Had you heard of the FISA Court before reading this chapter? Do you now have a better understanding of what it is?