CHAPTER 1

INTRODUCING BINARY AND HEXADECIMAL NUMBERS

I am ill at these numbers.

WILLIAM SHAKESPEARE
(1564–1616) in *Hamlet* (1601)

In this chapter we will learn about:

- Counting on fingers and toes
- Place-value number systems
- Using powers or exponents
- The binary number system
- The hexadecimal number system
- Counting in the binary and hexadecimal systems
- Using wires to represent numbers

Why Do We Need to Know this Stuff?

The number system with which we are most familiar is the *decimal system,* which is based on ten digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. As we shall soon discover, however, it's easier for electronic systems to work with data that is represented using the *binary* number system, which comprises only two digits: 0 and 1.

Unfortunately, it's difficult for humans to visualize large values presented as strings of 0s and 1s. Thus, as an alternative, we often use the *hexadecimal* number system, which is based on sixteen digits that we represent by using the numbers 0 through 9 and the letters A through F.

Familiarity with the binary and hexadecimal number systems is necessary in order to truly understand how computers and calculators perform their magic. In this chapter, we will discover just enough to make us dangerous, and then we'll return to consider number systems and representations in more detail in Chapters 4, 5, and 6.

Counting on Fingers and Toes

The first tools used as aids to calculation were almost certainly man's own fingers. It is no coincidence, therefore, that the word "digit" is used to refer to a finger (or toe) as well as a numerical quantity. As the need grew to represent greater quantities, small stones or pebbles could be used to represent larger numbers than could fingers and toes. These had the added advantage of being able to store intermediate results for later use. Thus, it is also no coincidence that the word "calculate" is derived from the Latin word for pebble.

Throughout history, humans have experimented with a variety of different number systems. For example, you might use one of your thumbs to count the finger joints on the same hand (1, 2, 3 on the index finger; 4, 5, 6 on the next finger; up to 10, 11, 12 on the little finger). Based on this technique, some of our ancestors experimented with base-12 systems. This explains why we have special words like *dozen,* meaning "twelve," and *gross,* meaning "one hundred and forty-four" ($12 \times 12 = 144$). The fact that we have 24 hours in a day (2×12) is also related to these base-12 systems.

Similarly, some groups used their fingers *and* toes for counting, so they ended up with base-20 systems. This is why we still have special

> **Note** Truth to tell, the first people to employ their fingers as counting aids didn't use the digits 0, 1, 2, and 3. Instead, they started counting at one, as in 1, 2, 3, 4, because if all one is doing is counting goats, for example, then the concepts of "zero goats" (and "negative goats") are relatively unimportant. In fact, it was not until around 600 AD that the use of zero as an actual value, along with the concept of negative numbers, first appeared in India.

words like *score,* meaning "twenty." However, due to the fact that we have ten fingers, the number system with which we are most familiar is the decimal system, which is based on ten digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9 (see note). The word *decimal* is derived from the Latin *decam,* meaning "ten." As this system uses ten digits, it is said to be *base-10* or *radix-10;* the term *radix* comes from the Latin word meaning "root."

> **Note** Dates in Western books have traditionally been shown in the form 1000 BC or 2001 AD, where BC stands for "Before Christ" and AD is an abbreviation of the Latin *Ano Domini,* meaning "Year of our Lord." In many cases, modern historians now prefer to use BCE, meaning "Before Common Era," instead of BC; and CE, meaning "Common Era," instead of AD. This preference is based on the fact that BCE and CE aren't associated with any particular religion. However, the BC/AD nomenclature is more familiar to nonhistorians, so that is what we are using in this book.

Place-Value Number Systems

Consider the concept of Roman numerals, in which I = 1, V = 5, X = 10, L = 50, C = 100, D = 500, M = 1,000, and so forth. Using this scheme, XXXV represents 35 (three tens and a five). One problem with this type of number system is that over time, as a civilization develops, it tends to become necessary to represent larger and larger quantities. This means that mathematicians either have to keep on inventing new symbols or start using lots and lots of their old ones. But the biggest disadvantage of this approach is that it's painfully difficult to work with (try multiplying CLXXX by DDCV and it won't take you long to discover what we mean).

An alternative technique is known as a *place-value system,* in which the value of a particular digit depends both on itself and on its position within the number. This is the way in which the decimal number system works. In this case, each column in the number has a "weight" associated with it, and the value of the number is determined by combining each digit with the weight of its column (Figure 1-1).

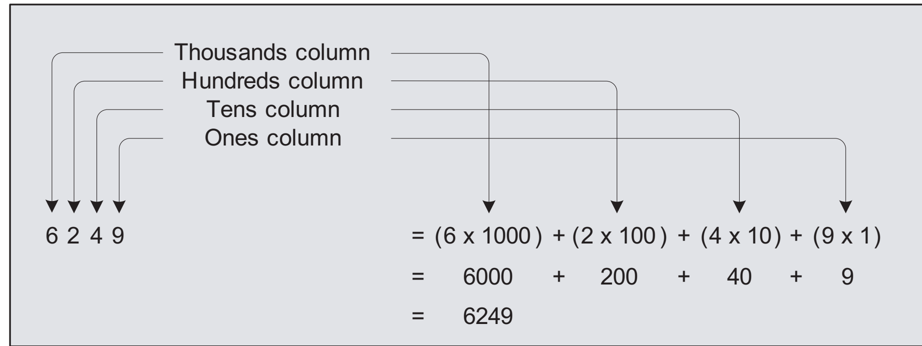

Figure 1-1. Combining digits and column weights in decimal.

Using Powers or Exponents

Another way of thinking about this is to use the concept of *powers;* for example, $100 = 10 \times 10$. This can be written as 10^2, meaning "ten to the power of two" or "ten multiplied by itself two times." Similarly, $1000 = 10 \times 10 \times 10 = 10^3$, $10{,}000 = 10 \times 10 \times 10 \times 10 = 10^4$, and so forth (Figure 1-2).

Rather than talking about using powers, some mathematicians prefer to refer to this type of representation as an *exponential form.* In the case of a number like 10^3, the number being multiplied (10) is known as the *base,* whereas the *exponent* (3) specifies how many times the base is to be multiplied by itself.

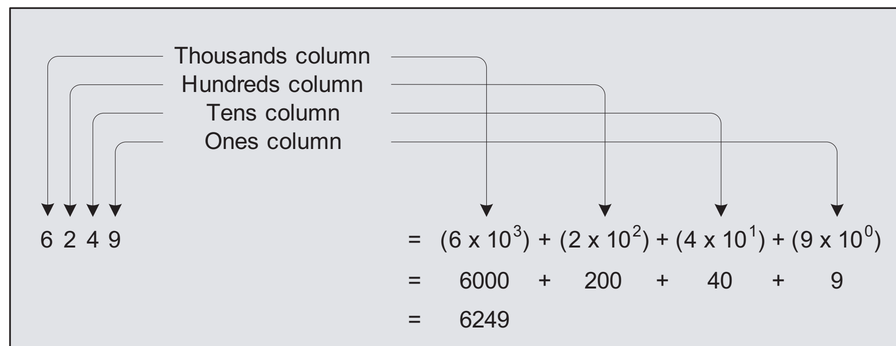

Figure 1-2. Using powers of ten.

There are a number of points associated with powers (or exponents) that are useful to remember:

- Any base raised to the power of 1 is the base itself, so $10^1 = 10$.
- Strictly speaking, a power of 0 is not really part of the series. By convention, however, any base to the power of 0 equals 1, so $10^0 = 1$.
- A value with an exponent of 2 is referred to as the *square* of the number; for example, $3^2 = 3 \times 3 = 9$, where 9 (or 3^2) is the square of 3.
- A value with an exponent of 3 is referred to as the *cube* of the number; for example, $3^3 = 3 \times 3 \times 3 = 27$, where 27 (or 3^3) is the cube of 3.
- The square of a *whole number* (0, 1, 2, 3, 4, etc.) is known as a *perfect square;* for example, $0^2 = 0$, $1^2 = 1$, $2^2 = 4$, $3^2 = 9$, $4^2 = 16$, $5^2 = 25$, and so on are perfect squares. (The concepts of whole numbers and their cousins are introduced in more detail in Chapter 4.)
- The cube of a *whole number* is known as a *perfect cube;* for example, $0^3 = 0$, $1^3 = 1$, $2^3 = 8$, $3^3 = 27$, $4^3 = 64$, $5^3 = 125$, and so on are perfect cubes.

But we digress. The key point here is that the column weights are actually powers of the number system's base. This will be of particular interest when we come to consider other systems.

Counting in Decimal

Counting in decimal is easy (mainly because we're so used to doing it). Commencing with 0, we increment the first column until we get to 9, at which point we've run out of available digits. Thus, on the next count we reset the first column to 0, increment the second column to 1, and continue on our way (Figure 1-3).

Similarly, once we've reached 99, the next count will set the first column to 0 and attempt to increment the second column. But the second column already contains a 9, so this will also be set to 0 and we'll increment the *third* column, resulting in 100, and so it goes.

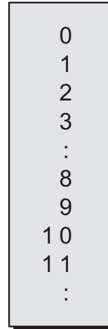

Figure 1-3. Counting in decimal.

The Binary Number System

Unfortunately, the decimal number system is not well suited to the internal workings of computers. In fact, for a variety of reasons that will become apparent as we progress through this book, it is preferable to use the binary (base-2) number system, which employs only two digits: 0 and 1.

Binary is a place value system, so each column in a binary number has a weight, which is a power of the number system's base. In this case we're dealing with a base of two, so the column weights will be $2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $2^3 = 8$, and so forth (Figure 1-4).

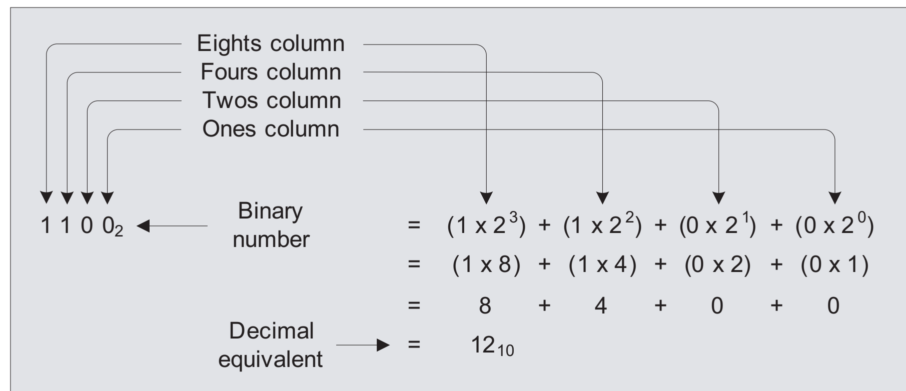

Figure 1-4. Combining digits and column weights in binary.

When working with number systems other than decimal, or working with a mixture of number systems as shown in Figure 1-4, it is common to use subscripts to indicate whichever base is in use at the time. For example, $1100_2 = 12_{10}$, which means $1100_{Binary} = 12_{Decimal}$.

Another common alternative is to prefix numbers with a character to indicate the base; for example, %1100, where the % indicates a binary value. (In fact there are a variety of such conventions, so you need to keep your eyes open and your wits about you as you plunge deeper into the mire.) Unless otherwise indicated, a number without a subscript or a prefix character is generally assumed to represent a decimal value.

Sometime in the late 1940s, the American chemist turned topologist turned statistician John Wilder Tukey realized that computers and the binary number system were destined to become increasingly important. In addition to coining the word "software," Tukey decided that saying "binary digit" was a bit of a mouthful, so he started to look for an alternative. He considered a variety of options, including *binit* and *bigit*, but eventually settled on *bit,* which is elegant in its simplicity and is used to this day.

Binary values of 1100_2 and 11001110_2 are said to be four and eight bits wide, respectively. Groupings of four bits are relatively common, so they are given the special name of *nybble* (or sometimes *nibble*). Similarly, groupings of eight bits are also common, so they are given the special name of *byte.* Thus, "*two nybbles make a byte,*" which goes to show that computer engineers do have a sense of humor (albeit not a tremendously sophisticated one).

Counting in binary

Counting in binary is even easier than counting in decimal; it just seems a little harder if you aren't familiar with it. As usual, we start counting from 0, and then we increment the first column to be 1, at which point we've run out of all the available digits. Thus, on the next count we reset the first column to 0, increment the second column to 1, and proceed on our merry way (Figure 1-5).

Similarly, once we've reached 11_2, the next count will set the first column to 0 and attempt to increment the second column. But the second column already contains a 1, so this will also be set to 0 and we'll increment the *third* column, resulting in 100_2. (Note that Figure 1-5 doesn't require us to use subscripts or special prefix symbols to indicate the binary values, because in this case they are obvious from the context.)

Binary	Decimal
0	0
1	1
1 0	2
1 1	3
1 0 0	4
1 0 1	5
1 1 0	6
1 1 1	7
1 0 0 0	8
:	:

Figure 1-5. Counting in binary.

Learning your binary "times tables"

Cast your mind back through the mists of time to those far-off days in elementary (junior) school. You probably remember learning your multiplication "tables" by rote, starting with the "two times table" ("one two is two, two twos are four, three twos are six, . . .") and wending your weary way onward and upward to the "twelve times table" ("ten twelves are one hundred and twenty, eleven twelves are one hundred and thirty-two, and—wait for it, wait for it—twelve twelves are one hundred and forty-four"). Phew!

The bad news is that we need to perform a similar exercise for binary: the good news is that, as illustrated in Figure 1-6, we can do the whole thing in about five seconds flat! That's all there is to it. This is the only binary multiplication table there is. This really is pretty much as complex as it gets!

Multiplication
0 x 0 = 0
0 x 1 = 0
1 x 0 = 0
1 x 1 = 1

Figure 1-6. The binary multiplication table.

Using Wires to Represent Numbers

Today's digital electronic computers are formed from large numbers of microscopic semiconductor switches called transistors, which can be turned on and off incredibly quickly (thousands of millions of times a second).

> **Note** Computers can be constructed using a variety of engineering disciplines, resulting in such beasts as electronic, hydraulic, pneumatic, and mechanical systems that process data using analog or digital techniques.
>
> For the purposes of this book however, we will consider only digital electronic implementations, because these account for the overwhelming majority of modern computer systems.

Transistors can be connected together to form a variety of primitive logical elements called *logic gates*. In turn, large numbers of logic gates can be connected together to form a computer.[1]

It's relatively easy for electronics engineers to create logic gates that can detect, process, and generate two distinct voltage levels.[2] For example, logic gates circa 1975 tended to use voltage levels of 0 volts and 5 volts. However, the actual voltage values used inside any particular computer are of interest only to the electronics engineers themselves. All we need know is that these two levels can be used to represent binary 0 and 1, respectively.

In the "counting in binary" example illustrated in Figure 1-5, there was an implication that our table could have continued forever. This is because numbers written with pencil and paper can be of any size, limited only by the length of your paper and your endurance. By comparison, the numbers inside a computer have to be mapped onto a physical system of logic gates and wires. For example, a single wire can be used to represent only $2^1 = 2$ binary values (0 and 1), two wires can be used to represent $2^2 = 2 \times 2 = 4$ different binary values (00, 01, 10, and 11), three wires can be used to represent $2^3 = 2 \times 2 \times 2 = 8$ differ-

[1]Transistors and logic gates are introduced in greater detail in the book *Bebop to the Boolean Boogie (An Unconventional Guide to Electronics)*, 2nd edition, by Clive "Max" Maxfield ISBN: 0-7506-7543-8.

[2]Voltage refers to electrical potential, which is measured in *volts*. For example, a 9-volt battery has more electrical potential than a 3-volt battery. Once again, the concept of voltage is introduced in greater detail in *Bebop to the Boolean Boogie (An Unconventional Guide to Electronics)*.

ent binary values (000_2, 001_2, 010_2, 011_2, 100_2, 101_2, 110_2, and 111_2), and so on.

The term *bus* is used to refer to a group of signals that carry similar information and perform a common function. In a computer, a bus called the *data bus* is, not surprisingly, used to convey data.

> **Note** In this context, the term "data" refers to numerical, logical, or other information presented in a form suitable for processing by a computer.For the purposes of this book however, we will consider only digital electronic implementations, because these account for the overwhelming majority of modern computer systems.
>
> Actually, "data" is the plural of the Latin *datum,* meaning "something given." The plural usage is still common, especially among scientists, so it's not unusual to see expressions like "*These data are . . .*"
>
> However, it is becoming increasingly common to use "data" to refer to a singular group entity such as information. Thus, an expression like "*This data is . . .*" would also be acceptable to a modern audience.

For the purposes of these discussions, let's assume that we have a data bus comprising eight wires that we wish to use to convey binary data. In this case, we can use these wires to represent $2^8 = 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 = 256$ different combinations of 0s and 1s. If we wished to use these different binary patterns to represent numbers, then we might decide to represent decimal values in the range 0 to 255 (Figure 1-7). For reasons that will be discussed in greater detail in Chapter 4, this form of representation is referred to as *unsigned binary numbers.*

In any numbering system, it is usual to write the most-significant digit on the left and the least-significant digit on the right. For example, when you see a decimal number such as 825, you immediately assume that the "8" on the left is the most-significant digit (representing eight hundred), whereas the "5" on the right is the least-significant digit (representing only five). Similarly, the most-significant binary digit, referred to as the *most-significant bit (MSB)*, is on the left-hand side of the binary number, whereas the *least-significant bit (LSB)* is on the right.

The Hexadecimal Number System

The problem with binary values is that, although we humans are generally good at making sense out of symbolic representations like words and numbers, we find it difficult to comprehend the meaning behind long strings of 0s and 1s. For example, the binary value 11001110_2 doesn't im-

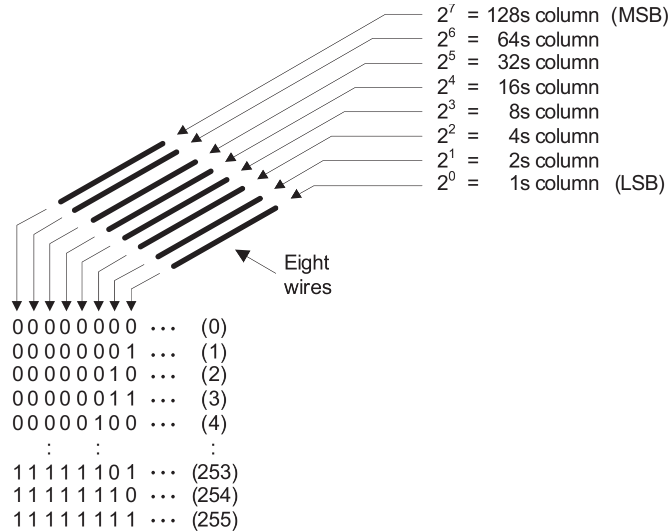

Figure 1-7. Using eight wires to represent unsigned binary numbers (values in parentheses are decimal equivalents).

mediately register with most of us, whereas its decimal equivalent of 206 is much easier to understand.

Unfortunately (as we shall come to see), translating values between binary and decimal can be a little awkward. However, any number system with a base that is a power of two (for example, 4, 8, 16, 32, etc.) can be easily mapped into its binary equivalent, and vice versa. In the early days of computing, it was common for computers to have data busses whose widths were wholly divisible by three (9 bits, 12 bits, 18 bits, and so forth). Thus, the octal (base-8) number system became very popular, because each octal digit can be directly mapped onto three binary digits (bits). More recently, computers have standardized on bus widths that are wholly divisible by eight (8 bits, 16 bits, 32 bits, and so forth). For this reason, the use of octal has declined and the hexadecimal (base-16) number system is now prevalent, because each hexadecimal digit can be directly mapped onto four bits.

As a base-16 system, hexadecimal requires 16 unique symbols, but inventing completely new symbols would not be a particularly efficacious solution, not the least that we would all have to learn them! Even worse, it would be completely impractical to modify every typewriter and com-

puter keyboard on the face of the planet. As an alternative, we use a combination of the decimal numbers 0 through 9 and the alpha characters A through F (Figure 1-8).

Counting in hexadecimal

The rules for counting in hexadecimal are the same as for any place-value number system. We commence with 0 and count away until we've used up all of our symbols (that is, when we get to F); then, on the next count, we set the first column to 0 and increment the second column (Figure 1-9).

Observe that the binary values in Figure 1-9 have been split into two 4-bit groups. This form of representation is not typical, but it is used here to emphasize the fact that each hexadecimal digit maps directly onto four bits.

Also note that binary and hexadecimal values are often prefixed (padded) with leading zeros. This practice is used to illustrate the number of digits that will be used to represent these values inside the computer. For our purposes here, we're assuming a data bus that's eight bits wide, so the values shown in Figure 1-9 have been padded to reflect this width.

Combining digits and column weights in hexadecimal

As with any place-value system, each column in a hexadecimal number has a weight associated with it; the weights are derived from the base. In this case we're dealing with a base of sixteen, so the column weights will be $16^0 = 1$, $16^1 = 16$, $16^2 = 256$, $16^3 = 4{,}096$, and so forth (Figure 1-10).

As we discussed earlier in this chapter, it's common practice to use subscripts to indicate whichever base is in use at the time. For example, $4F0A_{16} = 20{,}234_{10}$, which means $4F0A_{Hexadecimal} = 20{,}234_{Decimal}$. As we

Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hexadecimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Figure 1-8. The sixteen hexadecimal digits.

Value	Decimal	Binary	Hexadecimal
zero	0	0000 0000	00
one	1	0000 0001	01
two	2	0000 0010	02
three	3	0000 0011	03
four	4	0000 0100	04
five	5	0000 0101	05
six	6	0000 0110	06
seven	7	0000 0111	07
eight	8	0000 1000	08
nine	9	0000 1001	09
ten	10	0000 1010	0A
eleven	11	0000 1011	0B
twelve	12	0000 1100	0C
thirteen	13	0000 1101	0D
fourteen	14	0000 1110	0E
fifteen	15	0000 1111	0F
sixteen	16	0001 0000	10
seventeen	17	0001 0001	11
eighteen	18	0001 0010	12
:	:	:	:

Figure 1-9. Counting in hexadecimal.

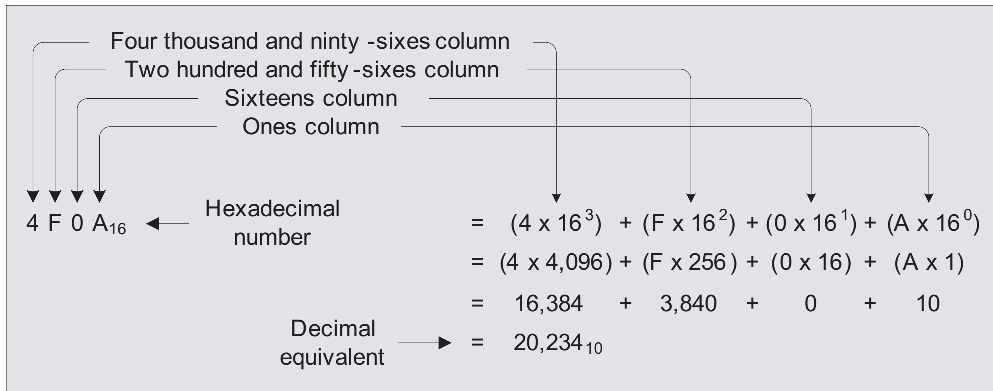

Figure 1-10. Combining digits and column weights in hexadecimal.

also noted, another common alternative is to prefix such numbers with a character to indicate the base. For example, $4F0A, where the $ indicates a hexadecimal value. (By now it may not surprise you to learn that, once again, there are a variety of such conventions.) As usual, any number without a subscript or a prefix character is generally assumed to represent a decimal value unless otherwise indicated.

Mapping between hexadecimal and binary

Mapping between hexadecimal and binary is extremely simple. For example, in order to convert the hexadecimal value $4F0A into binary, all we have to do is to replace each hexadecimal digit with its 4-bit binary equivalent. That is,

$4 → %0100, $F → %1111, $0 → %0000, and $A → %1010

So $4F0A in hexadecimal equates to %0100111100001010 in binary.

Similarly, it's easy to convert a binary number such as %1100011010110010 into its hexadecimal equivalent. All we have to do is to split the binary value into 4-bit nybbles and to map each nybble onto its corresponding hexadecimal digit. That is,

%1100 → $C, %0110 → $6, %1011 → $B, and %0010 → $2

So %1100011010110010 in binary equates to $C6B2 in hexadecimal.

Review

This chapter introduced the concepts of place-value number systems, the binary and hexadecimal number systems, using powers or exponents, and using wires to represent numbers. Now, just to make sure that we fully understand these concepts, let's answer the following questions and/or perform the following tasks:

1) What is the decimal equivalent of the Roman numeral MMMDCCLXXVIII?
2) In the case of a value such as 10^3, what terms are used to refer to the 10 and the 3, respectively?
3) What term is used to refer to a binary digit, and to what do the terms *nybble* and *byte* refer?

4) What are two different ways to indicate that the value 10100101 is a binary number?
5) What are two different ways to indicate that the value 10 is a hexadecimal number?
6) Convert the decimal value 125_{10} into an 8-bit unsigned binary equivalent.
7) Convert the binary value 10100101_2 into its hexadecimal equivalent.
8) Convert the hexadecimal value $E6 into its binary equivalent.
9) How many digits would there be in a quinary (base-5) place-value number system (list these digits); what would be the first five column weights in such a system; and how would one count in such a system?
10) Assuming positive numbers starting at 0, what range of decimal numbers could be represented using the different binary patterns that could be accommodated by a set of seven wires? How about nine wires?

Bonus Question Cast your mind back to our discussions on counting with wires and Figure 1-7. There are two key problems associated with this form of binary representation. Can you spot what they are?

Answer: The first problem is that eight wires allow us to represent only 256 different combinations of 0s and 1s. For the purposes of these discussions, we decided to use these binary patterns to represent the decimal values 0 to 255, but this is obviously very restrictive. What happens if we wish to represent values greater than 255?

The second problem is that the scheme we demonstrated here allows us to represent only positive values. What happens if we wish to represent both positive and negative values?

Both of these issues are addressed in Chapter 4. However, it would be useful for you to start thinking about them now and see if you can come up with your own solutions. Later, when we reach Chapter 4, you can compare your ideas with the techniques that have been developed by computer scientists and see how well you did.