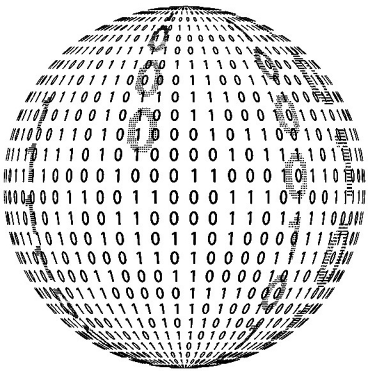
CHAPTER 9

Computer Engineering



9.1 INTRODUCTION

Why do cars, trucks, planes, ships, and almost everything else today contain computers? After all, until the 1960s we got along perfectly well without computers. As late as 1965, the question, “Should a car or truck or plane or ship or anything else contain a computer?,” would have seemed ridiculous. In those days, a computer was not only much more expensive than a car, but also bigger.

9.2 MOORE’S LAW

In the late 1960s, the integrated circuit was invented, and computers began to shrink in both size and cost. In the 1990s it was also asserted that “if a Cadillac had shrunk in size and cost as fast as a computer did since 1960, you could now buy one with your lunch money and hold it in the palm of your hand.” The basic message, however, was, and remains, valid. From the 1960s until the present, integrated circuits, the building blocks of computers, have doubled their computing power every year or two. This explosive progress is the result of a technological trend called *Moore’s Law* that states that the number of transistors on an integrated circuit “chip” will double every year or two. The name honors electronics pioneer Gordon Moore, who proposed it in 1965. Moore’s Law is not a law of nature, but an empirical rule of thumb, and it has held true for four and a half decades. Many people (including Gordon Moore) expect Moore’s law will end by around 2025 or 2030.

Today's digital computers are so small that, in most applications, size is no longer an issue. For example, a computer for controlling the air/fuel mixture in an automobile is typically housed along with its power supply and communication circuitry in an assembly about the size of a small book. As a result, people reliably control not just the hundreds of horsepower of an automobile, but the delivery of energy routinely available to them in modern industrial societies while minimizing pollution emitted in the course of energy conversion.

The subjects of control and binary logic have taken on a life of their own in the form of computers and other information systems. Indeed, such systems have become so pervasive as to lead many people to assert that the industrialized world changed in the late 20th century from an industrial society to an information society.

The term **digital** can apply to any number system, such as the base 10 system used in ordinary arithmetic. However, today's computers are based on a simpler number system: the base two or **binary** system. Therefore, this chapter focuses on binary logic and computation.

Computation was implemented first by mechanical devices, then by **analog computers**, and today by **digital computers**. A particularly convenient way to use computation to accomplish control is through the use of **binary logic**. A means of summarizing the results of binary logic operations is through **truth** tables that can be expressed as **electrical logic circuits**. This technique is useful not only in digital control, but also in other areas of computation, such as **binary arithmetic** and **binary codes**. Binary arithmetic and information are the basis of computer software. Binary logic also enables us to define the engineering variable *information.* In this chapter these topics will be mostly illustrated by reference to a familiar automotive application: deciding when a seat belt warning light should be turned on.

As an afterword, the process by which actual computers do these things, the **hardware–software connection**, will be summarized.

Millions of *embedded* computers (that is, computers placed within other systems and dedicated to serving those systems) now can be found in thousands of applications. Electronic and computer engineers have the job of reproducing this computer population, which brings forth a new generation every two or three years. Some develop the electronic circuits that are the basis of electronic computation. Others devise new computer architectures for using those circuits. Yet others develop the "software," which are the instructions that make computers perform as intended. Many other engineers apply these tools to everything from kitchen appliances to space vehicles.

9.3 ANALOG COMPUTERS

For hundreds of years, engineers proved ingenious and resourceful at using mechanical devices to control everything from the rotation of water wheels and windmills to the speed of steam engines and the aiming of guns on battleships. However, such mechanical systems had major disadvantages. They were limited in their speed and responsiveness by the mechanical properties of the components and they had to be custom designed for every control challenge.

By the 20th century, this had led engineers to seek more flexible, responsive, and general types of controls. As a first step, engineers put together standardized packages of springs, wheels, gears, and other mechanisms. These systems also proved capable of solving some important classes of mathematical problems defined by differential equations. Because these collections of standard mechanisms solved the problems by creating a mechanical analogy for the equations, they were called analog computers. Bulky and inflexible, they often filled an entire room and were "programmed" by a slow and complex process of manually connecting wires and components in order to do a computation. Despite these drawbacks, in such applications as predicting the tides, determining the performance of electrical transmission lines, or designing automobile suspensions, analog computers marked a great advance over previous equation-solving methods.

9.4 FROM ANALOG TO DIGITAL COMPUTING

Meanwhile, a second effort, underway since about 1800, sought to calculate solutions numerically using arithmetic done by people. Until about 1950, the word *computer* referred to a person willing to calculate for wages. Typically, these were selected members of the workforce whose economic status forced them to settle for relatively low pay. It was not until about 1900 that these human computers were given access to mechanical calculators and not until the 1920s that practical attempts at fully automated mechanical calculations were begun.

Meanwhile, as early as the 1820s, beginning with the ideas of the British scientist Charles Babbage, attempts had been made to do arithmetic accurately using machinery. Because these proposed machines operated on digits as a human would (rather than forming analogies), they were called *digital* computers. Digital techniques were also used to control machinery. For example, the French inventor Joseph Marie Jacquard (1752–1834) used cards with holes punched in them as a digital method to control the intricate manipulations needed to weave large complex silk embroideries.

Humans learn digital computing using their 10 fingers.[1] Human arithmetic adapted this 10-digit method into the decimal system. Digital computers can be built on a decimal basis, and some of the pioneers, such as Babbage in the 1840s and the team at the University of Pennsylvania who, in the 1940s built a very early electronic digital computer, the ENIAC, adopted this decimal system.

[1]The word *digitus* is literally "finger" in Latin, indicating the discrete nature of such counting schemes.

9.5 BINARY LOGIC

However, computer engineers quickly found that a simpler system less intuitive to humans proved much easier to implement with electronics. This is the binary system, based on only two digits: one and zero. The Jacquard loom was such a binary system, with a hole in a card representing a 1 (one) and the lack of a hole representing 0 (zero). In other applications, turning on a switch might represent a 1, and turning it off might represent a 0. The logic and mathematics of this system were developed mainly by the British mathematician George Boole (1815–64), whose contributions were so important that the concept is often referred to as **Boolean algebra**.

Binary logic begins with statements containing variables symbolized by letters such as X or Y. The statement can assert anything whatsoever, whether it is "The switch is open" or "There is intelligent life on a planet circling the star Procyon." The variable representing the statement can be assigned either of two values, 1 or 0, depending on whether the statement is true or false. (We will adopt the convention of using the value 1 for true statements and the value 0 for false ones.)

Binary logic permits three, and only three, operations to be performed, **AND**, **OR**, and **NOT:**

AND (sometimes called "intersection" and indicated by the symbol • or *) means that, given two statements X and Y, if both are true, then $X \bullet Y = 1$. If either one is false, then $X \bullet Y = 0$. For example, the statement "It is raining (X) and the sun is out (Y)" is true only if both it is raining *and* the sun is out.

OR (sometimes called "union" and indicated by the symbol $+$) the inclusive "or" means that, given two statements X and Y, if either one or both are true, then $X + Y = 1$, while only if both are false is $X + Y = 0$. For example, the statement "It is raining (X) *or* the sun is out (Y)" is true in all cases except when *both* are not true.

NOT (sometimes called "negation" and indicated here by the postsymbol ′ as in X', but sometimes indicated in other texts with an overbar as in $\dot{X}$) is an operation performed on a single statement. If X is the variable representing that single statement, then $X' = 0$ if $X = 1$, and $X' = 1$ if $X = 0$. Therefore, if X is the variable representing the statement "It is raining," then X' is the variable representing the statement "It is not raining."

These three operations provide a remarkably compact and powerful tool kit for expressing any logical conditions imaginable. A particularly important type of such a logical condition is an "if–then" relationship, which tells us that **if** a certain set of statements has some particular set of values, **then** another related statement, often called the **target statement**, has some particular value.

Consider, for example, the statement "*If a cold front comes in from the south or the air pressure in the north remains constant, but not if the temperature is above 50°F, then it will rain tomorrow.*" The target statement and each of the other statements can be represented by a variable. In our example, X can be the target statement "It will rain tomorrow," and the three other statements can be expressed by $A =$ "a cold front comes in from the south," $B =$ "the air pressure in the north is constant," and $C =$ "the temperature is above 50°F". The connecting words can be expressed using their symbols. Thus, this long and complicated sentence can be expressed by the short and simple **assignment** statement:

$$X = A + B \cdot C'$$

This is not a direct *equivalence* in which information flows both ways across the equation. Specifically, in an *assignment,*[2] the information on the right-hand side is assigned to X but not vice-versa. This distinction is required because of the way that computers actually manipulate information.

However, as written above, the statement still presents a problem. In which order do you evaluate the operations? Does this make a difference? A simple example will show that it *does* make a difference! Consider the case $A=1$, $B=0$, $C=1$. Let the symbol + is used first, the symbol • next, and the symbol ′ last. Then in the preceding statement, $A+B=1+0=1$ and $(A+B) \bullet C = 1 \bullet C = 1 \bullet 1 = 1$. But since $C' = 1' = 0$, then $X=0$. But if the symbols are applied in the reverse order (′ first, • next, and + last), then $C' = 1' = 0$, $B \bullet C' = 0 \bullet 1 = 0$, $A + B \bullet C' = 1 + 0 = 1$, and $X = 1$.

So, to get consistent results when evaluating logic statements, a proper order must be defined. This is similar to standard precedence rules used in arithmetic. That order is defined as follows:

1. All NOT operators must be evaluated **first**, then
2. all AND operators (starting from the right if there is more than one) **second**, and
3. all OR operators (starting from the right if there is more than one) are evaluated **third.**

If a different order of operation is desired, that order must be enforced with parentheses, with the operation within the innermost remaining parentheses being evaluated first, after which the parentheses is removed. In our example above, the proper answer with explicit parentheses would have been written $X = A + (B \bullet C')$ and evaluated as $X = 1 + (0 \bullet 1') = 1$. However, the value of the expression $X = (A+B) \bullet C'$ is $X = (1+0) \bullet 1' = 1 \bullet 1' = 1 \bullet 0 = 0$.

EXAMPLE 9.1

Consider the following statement about a car: “The seat belt warning light is on.” Define the logic variable needed to express that statement in binary logic.

Need:	Logic variable (letter) = “ …” where the material within the quotes expresses the condition under which the logic variable has the value 1 = true and 0 = false.
Know–How:	Choose a letter to go on the left side of an equivalence. Express the statement in the form it would take if the content it referred to is true. Put the statement on the right side in quotes.
Solve:	W = “the seat belt warning light is on.”

This example is trivially simple, but more challenging examples can arise quite naturally. For example, if there are two or more connected constraints on a given action, then the methods of Boolean algebra are surefire ways of fully understanding the system in a compact way.

[2]In some programming languages such logic statements are written with an *assignment* command, “:=” and not just an “=” command so that our Boolean statement could be written as $X := A + B \cdot C'$ to reinforce the fact that these are not reversible equalities.

EXAMPLE 9.2

Consider the statement involving an automobile cruise control set at a certain speed (called the "set speed"): "Open the throttle if the speed is below the set speed and the set speed is not above the speed limit." Express this as a logic formula, and evaluate the logic formula to answer the question: "*If the speed is below the set speed and the set speed is above the speed limit, then will the throttle be opened*?" Answer using these variables: the car's speed is 50. miles per hour, the set speed is 60. miles per hour, and the speed limit is 45 miles per hour.

Need: A binary logic formula expressing the statement "Open the throttle if the speed is below the set speed and the set speed is not above the speed limit," and an evaluation of the formula for the situation when the speed is 50. miles per hour, the set speed is 60. miles per hour, and the speed limit is 45 miles per hour.

Know: Any statement capable of being true or false can be represented by a variable having values 1 = true and 0 = false, and these variables can be connected by AND (•), OR (+), and NOT (′).

How: Define variables corresponding to each of the statements, and use the three connectors to write a logic formula.

Solve: Let X = "throttle is open."
Let A = "speed is below set speed."
Let B = "set speed is above speed limit."

Then the general logic formula expressing the target statement is $X = A \bullet B'$

If the speed is 50. miles per hour, and the set speed is 60. miles per hour, then $A = 1$. If the set speed is 60. miles per hour, and the speed limit is 45 miles per hour, then $B = 1$. Substituting these values, the general logic formula gives $X = 1 \bullet 1'$. Evaluating this in the proper order gives $X = 1 \bullet 0 = 0$.

In plain English, if the speed is below the set speed, and the set speed is above the control speed limit, the throttle will not open.

EXAMPLE 9.3

Suppose we want to find a Boolean expression for the truth of the statement "W = the seat belt warning light should be on in my car," using all of the following Boolean variables:

Case 1:

D is true if the driver seat belt is fastened.
Pb is true if the passenger seat belt is fastened.
Ps is true if there is a passenger in the passenger seat.

Case 2:
For actuation of the warning light, include the additional Boolean variable: M is true if the motor is running.

Need: $W = ?$

Know–How: Put the W variable on the left side of an assignment sign W = and then array the variables on the other side of the assignment sign.

Case 1

(a) Put D, Ps, and Pb on the right side of the assignment sign. Thus, the temporary (for now, incorrect) assignment statement is $W = D\ Ps\ Pb$.

(b) Connect the variables on the right side with the three logic symbols •, +, and ′ so that the relationship among the variables on the right side correctly represents the given statement.

A good way to do this is to simply put the symbols the way they should appear in the if statements. For example, since part of the if statement Pb contains the words "the passenger's seat belt is fastened," it should also contain a "not." The corresponding logic variable will appear as Pb'.

Also, you only care about the passenger's seat belt if the passenger is sitting in the seat, that is, the intersection between these two variables.

Solve: A solution in English is: "if the driver's seat belt is not fastened" or "if there is a passenger in the passenger seat" and "the passenger's seat belt in not fastened", then "the seat belt warning light should be on in my car", or $W = D' + Ps \cdot Pb'$.

Once written, a logic equation can now be solved for any particular combination of variables. This is done by first plugging in the variable and then carrying out the indicated operation.

Case 2
For activation of the warning light, the light will go on if the motor is running, and if either the driver's seat belt is not fastened or if there is a passenger in the passenger seat and that seatbelt is not fastened. This is written as:

$$W = M \cdot \left(D' + Ps \cdot Pb'\right)$$

9.6 TRUTH TABLES

It is often convenient to summarize the results of a logic analysis for all possible combinations of the values of the input variables of an "if … then" statement. This can be done with a truth table**.** It is simply a table with columns representing variables and rows representing combinations of variable values. The variables for the "if" conditions start from the left, and their rows can be filled in systematically to include *all possible combinations* of inputs. The column at the far right represents "then." Its value can be computed for each possible input combination.

EXAMPLE 9.4

Consider the condition in Example 9.2: open the throttle if the speed is below the set speed and the set speed is not above the speed limit. The "**if**" conditions are A = "speed is below the speed limit" and B = "set speed is above the speed limit." The "**then**" condition is X = "open the throttle." The truth table is set up as shown here.

A	B	B'	$X = A \bullet B'$

Need: All 16 entries to the truth table.
Know: Negation operator, $'$ and the AND operator •.
How: Fill in all possible binary combinations of statements A and of B.

[3]If the input had three columns, A, B, and C, you would count from 0 to 7 in binary (000 to 111). If four columns, the 16 entries would count from 0 to 15 in binary (000 to 1111), and so on.

Solve: One convenient way of making sure you insert all the possible input values is to "count" in binary from all zeroes at the top to all ones at the bottom. (If you're not already able to count in binary, this is explained on the next page or so.) In this example, the top line on the input side represents the binary number 00 (equal to decimal 0), the second line is 01 (decimal 1), the third is 10 (decimal 2), and the fourth is 11 (decimal 3).[3]

A	B	B'	$X = A \bullet B'$
0	0		
0	1		
1	0		
1	1		

Next the value of the "then" ("open the throttle") is computed for each row of inputs (this is done here in two steps, first computing B', then computing $X = A \bullet B'$).

A	B	B'	$X = A \bullet B'$
0	0	1	0
0	1	0	0
1	0	1	1
1	1	0	0

In English, this truth table is telling us that the only condition under which the control will open the throttle ($X = 1$) is when both the speed is below the set speed and the set speed is below the speed limit. This is the way we *should* want our cruise control to operate!

Truth tables can also be conveniently expressed as electric circuits. Indeed, this capability is the essence of computing. This capability is further explored in the exercises.

9.7 DECIMAL AND BINARY NUMBERS

In the decimal or base 10 number system, digits are written to the left or right of a dot called the **decimal point** to indicate values greater than one or less than one. Each digit is a **placeholder** for the next power of 10. The digits to the left of the decimal point are whole numbers, and as you move to the left every number placeholder increases by a factor of 10. On the right of the decimal point the first digit is **tenths** (1/10), and as you move further right every number placeholder is 10 times smaller (see Fig. 9.1).

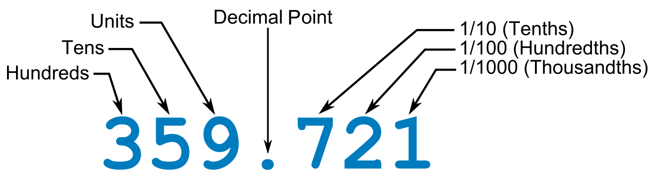


FIGURE 9.1

The structure of a decimal number.

The number **6357.** has four digits to the left of the decimal point, with "7" in the **units** place, "5" in the **tens** place, "3" in the **hundreds** place and 6 in the **thousands** place. You can also express a decimal number as a whole number plus tenths, hundredths, thousandths and so forth. For example, in the number 3.76, the 3 to the left of the decimal point is the "whole" number, the 7 on the right side of the decimal point is in the "tenths" position, meaning "7 tenths" or 7/10, and the 6 is in the hundredths position. So, 3.76 can be read as: "3 and 7 tenths and 6 hundredths".

There is nothing that requires us to have 10 different digits in a number system. The **base-10** number system probably developed because we have 10 fingers, but if we happened to have eight fingers instead, we would probably have a base-8 number system. In fact, you can have a **base-anything** number system. There are often reasons to use different number bases in different situations.

The binary[4] (base 2) number system is similar to the decimal system in that digits are placed to the left or right of a "point" to indicate values greater than one or less than one (see Fig. 9.2). For binary numbers, the first digit to the left of the "binary point" is called the **units**. As you move further to the left of the binary point, every placeholder increases by a factor of **2**. To the right of the binary point the first digit is **half** (1/2), and as you move further to the right every placeholder number becomes **half again smaller**. Table 9.1 below shows a few equivalent decimal and binary whole numbers.

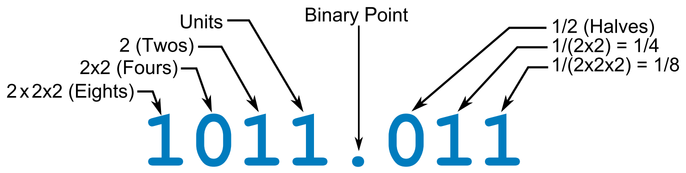


FIGURE 9.2

The structure of a binary number.

Table 9.1 Decimal and Binary Numbers.

Decimal:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Binary:	0	1	10	11	100	101	110	111	1000	1001	1010	1011	1100	1101	1110	1111

So, how do you convert binary numbers to decimal numbers? Several examples are given below to show the steps in convert several binary numbers to decimal numbers.

1. **What is 1111 in decimal?**
   - The "1" in the leftmost position is in the "2×2×2" (2^3) position, so that means $1\times2\times2\times2=8$
   - The next "1" is in the "2×2" (2^2) position, so that means $1\times2\times2=4$
   - The next "1" is in the "2" (2^1) position, so that means $1\times2=2$
   - The last "1" is in the units ($2^0=1$) position, so that means $1\times1=1$
   - Answer: $1111=8+4+2+1=15$ in decimal.
2. **What is 1001 in decimal?**
   - The "1" in the leftmost position is in the "2×2×2" (2^3) position, so that means $1\times2\times2\times2=8$

[4]The word **binary** comes from "bi-" meaning two. We use it in words such as "bicycle" (two wheels) and "binocular" (two eyes).

- The next "0" is in the "2 × 2" (2^2) position, so that means 0 × 2 × 2 = 0
- The next "0" is in the "2" (2^1) position, so that means 0 × 2 = 0
- The last "1" is in the units (2^0) position, so that means 1 × 1 = 1
- Answer: 1001 = 8 + 0 + 0 + 1 = 9 in decimal

3. **What is 1.1 in decimal?**
   - The "1" on the left side of the binary point is in the units (2^0) position, so that means 1 × 1 = 1
   - The 1 on the right side is in the "halves" (2^{-1}) position, so that means 1 × (1/2) = 0.5
   - So, 1.1 is "1 and 1 half" = 1.5 in decimal
4. **What is 10.11 in decimal?**
   - The "1" in the leftmost position is in the "2" (2^1) position, so that means 1 × 2 = 2
   - The "0" is in the units (2^0) position, so that means 0 × 1 = 0
   - The first "1" on the right of the point is in the "halves" (2^{-1}) position, so that means 1 × (1/2) = 0.50
   - The last "1" on the right side is in the "quarters" (2^{-2}) position, so that means 1 × (1/4) = 0.25
   - So, 10.11 is 2 + 0 + 1/2 + 1/4 = 2.75 in decimal

A single binary digit (0 or 1) is called a "**bit**" For example, the binary number 11,010 has five bits. The word **bit** is made from the words "**b**inary dig**it**" Bits are usually combined into 8-bit collections called **byte**. With an 8-bit byte, you can represent 256 values ranging from 0 to 255, as shown below:

```
0 = 00000000
1 = 00000001
2 = 00000010
................
254 = 11111110
255 = 11111111
```

Bytes often come with **prefixes** like kilo, mega, and giga, as in kilobyte, megabyte, and gigabyte. Table 9.2 lists the actual sizes of these binary numbers.

Table 9.2 Binary Number Byte Prefixes.

Name	Size
Kilo (K)	2^{10} = 1024
Mega (M)	2^{20} = 1,048,576
Giga (G)	2^{30} = 1,073,741,824
Tera (T)	2^{40} = 1,099,511,627,776
Peta (P)	2^{50} = 1,125,899,906,842,624
Exa (E)	2^{60} = 1,152,921,504,606,846,976
Zetta (Z)	2^{70} = 1,180,591,620,717,411,303,424
Yotta (Y)	2^{80} = 1,208,925,819,614,629,174,706,176

A terabyte hard drive actually stores 10^{12} bytes.[5] How could you possibly need a terabyte of disk space? When you consider all the digital media available today (music, games, and video), it is not difficult to fill a terabyte of storage space. Terabyte storage devices are fairly common, and indeed there are some petabyte storage devices available.

9.8 BINARY ARITHMETIC

The value of a **bit** depends on its position relative to the "binary point." For example, the binary number 11,010.101 has a decimal value computed from the second and third rows of Table 9.3 as $\mathbf{16}\times \mathit{1}+\mathbf{8}\times \mathit{1}+\mathbf{4}\times \mathit{0}+\mathbf{2}\times \mathit{1}+\mathbf{1}\times \mathit{0}+\mathbf{0.500}\times \mathit{1}+\mathbf{0.250}\times \mathit{0}+\mathbf{0.125}\times \mathit{1}=\mathbf{26.625}$.

Table 9.3 Converting the Binary Number 11010.101 to a Decimal Number.

Placeholder	2^4	2^3	2^2	2^1	2^0	**.**	2^{-1}	2^{-2}	2^{-3}
Bit	1	1	0	1	0	.	*1*	*0*	*1*
Decimal value	16	8	4	2	1	.	1/2 = 0.500	1/4 = 0.250	1/8 = 0.125

Let's begin with the Rules of Binary Addition.

$0+0=0$
$0+1=1$
$1+0=1$
$1+1=10$, so carry the 1 to the next bit and save the 0

For example, adding 010 (digital 2) to 111 (digital 7) gives:

$$\begin{array}{r} 010 \\ +\,111 \\ \hline 1001 \end{array} \quad \text{(digital 9)}$$

Binary addition is conceptually identical to decimal addition. However, instead of carrying powers of 10, one carries powers of two. Here are the formalized steps to follow for the carry digits:

1. Starting at the right, $0+1=1$ for the first digit (No carry needed)
2. The second digit is $1+1=10$ for the second digit, so save the 0 and carry the 1 to the next column
3. For the third digit, $0+1+1=10$, so save the zero and carry the 1
4. The last digit is $0+0+1=1$

[5]The capacities of computer storage devices are typically advertised using their SI standard values, but the capacities reported by software operating systems uses the binary values. The standard SI terabyte (TB) contains 1,000,000,000,000 bytes $=10{,}00^4$ or 10^{12} bytes. However, in binary arithmetic, a terabyte contains 1,099,511,627,776 bytes $=10{,}24^4$ or 2^{40} bytes.

5. So the answer is 1001 (digital 9 - you can see it is correct since decimal 2+7=9)

Binary subtraction is conceptually identical to decimal subtraction. However, instead of borrowing powers of 10, one borrows powers of two.

Rules of Binary Subtraction

$0-0=0$
$0-1=1$, and borrow 1 from the next more significant bit
$1-0=1$
$1-1=0$

The following examples illustrate "borrowing" in binary subtraction.

$$\begin{array}{r} 10 \\ -\,1 \\ \hline 1 \end{array} \qquad \begin{array}{r} 100 \\ -\,10 \\ \hline 10 \end{array} \qquad \begin{array}{r} 1010 \\ -\,110 \\ \hline 100 \end{array}$$

Can you complete Table 9.4? You can self-check against their decimal equivalents.

Table 9.4 Simple Binary Arithmetic Examples.

Example 1		Example 2		Practice 1		Practice 2	
Binary	Decimal	Binary	Decimal	Binary	Decimal	Binary	Decimal
1001	9	1001	9	1011		1011	
+101	+5	–101	–5	+110		–110	
1110	14	100	4				

The process of binary subtraction may be viewed as the addition of a negative number. For example, 3–2 may be viewed as 3+(–2). To do this you must determine the negative representation of a binary number. One way of doing this is with the **one's complement**.

The one's complement of binary number is found by changing all the ones to zeroes and all the zeroes to ones as shown below:

Number	One's Complement
10,011	01100
101,010	010101

To subtract a smaller number from a larger number using the one's complement method you:

1. Determine the one's complement of the smaller number,
2. Add the one's complement to the larger number,
3. Remove the final carry and add it to the result (this step is called the "end-around carry").

EXAMPLE 9.5

Do the following subtraction: 11001 (decimal 25)–10011 (decimal 19).

Need: 11001–10011 = ______? (a binary number)

Know–How: Step1: The one's complement of 10,011 is 01,100

Step2: Adding the one's complement to the larger number gives 01100 + 11001 = 100101

Step3: Removing the final carry and adding it to the result gives 00101 + 1 = 00,110

Solve: 11,001 – 10,011 = 110. To verify that this is correct, convert each base 2 number to decimal and repeat the subtraction, or **25 – 19 = 6**.

To subtract a larger number from a smaller number, the one's complement method is as follows:

1. Determine the one's complement of the larger number,
2. Add the one's complement to the smaller number (the result is the one's complement of the answer),
3. Take the one's complement of the result to get the final answer. Don't forget to add the minus sign since the result is negative.

EXAMPLE 9.6

Do the following subtraction: 1001 (decimal 9)–1101 (decimal 13).

Need: 1001–1101 = ? (a binary number)

Know–How: Step1: The one's complement of the larger number 1101 is 0010

Step2: Adding the one's complement to 1001 gives 0010 + 1001 = 1011

Step3: Add a minus sign to the one's complement of 1011 to get 1001–1101 = -0100

Solve: 1001 – 1101 = –100. To verify that this is correct, convert each base 2 number to decimal and repeat the subtraction, or **9 – 13 = –4**.

The rest of the familiar arithmetic functions can also be carried out in binary. **Fractions** can be expressed in binary by means of digits to the right of a binary point. Once again, powers of 2 take the role that powers of 10 play in digital arithmetic. Thus, the decimal fraction 0.5 (i.e., ½) is the binary fraction 0.1 and the decimal fraction ¼ is the binary fraction 0.01, and so on. A fraction that is not an even power of 1/2 can be expressed as a sum of binary numbers. Thus, the decimal ⅜ = 0.0011 in binary, which is the sum of decimal ¼ + ⅛.

Multiplication and **division** can be carried out using the same procedures as in decimal multiplication and "long division." The only complication is, again, systematically "carrying" and "borrowing" in powers of two, rather than powers of 10. Here we only give the rules for binary multiplication.

Rules for Binary Multiplication

$0 \times 0 = 0$

$0 \times 1 = 0$

$1 \times 0 = 0$

$1 \times 1 = 1$, and no carry or borrow bits

Rules for Binary Division.

The good news is that binary division is a little easier than decimal division because instead of having to guess how many times the divisor fits into the dividend, in binary division the answer will either be 0 or 1. But because division involves multiplication and subtraction operations which may also involve "borrowing" from the next digit, binary division is somewhat more complicated than binary multiplication. For example, suppose we want to divide 15 by 5 in binary. Since 5 in binary is 101 and 15 in binary is 1111, their binary division looks like this:

$$\begin{array}{r} 11 \\ 101\overline{)1111} \\ -1010 \\ 0101 \\ -101 \\ 0 \end{array}$$

So, the answer is 11 (or 3in decimal).

EXAMPLE 9.7

If a powerful race car has an air to fuel ratio (*A*/*F*) of 15 [kg air]/[kg fuel] and the air intake draws in 1.5 kg of air per second, how much fuel must be injected every second? Solve in binary to five significant binary digits.

Need: Fuel rate = ______ kg fuel/s in binary?

Know–How: Fuel flow rate = (*F*/*A*) × 1.5 kg air/s = (1/15) [kg fuel]/[kg air] × 1.5 [kg air/s] = 0.10 [kg fuel/s].

To illustrate binary arithmetic, let's break down this problem into two separate ones. While unnecessary to do this, one will illustrate binary division and binary multiplication.

The division problem will be 1/15 (decimal) = 1/1111 (binary), and the multiplication problem will take the solution of that problem and then multiply it by 1.5 (decimal) = 1.1 (binary).

Solve: Start with 1/1111, then binary multiply that answer by 1.1.

$$\begin{array}{r} .00010001 \\ 1111\overline{)1.00000000} \\ \underline{1111} \\ 10000 \\ \underline{1111} \\ 1 \end{array} \qquad \begin{array}{r} 0.00010001 \\ \underline{\times 1.1} \\ 0.00010001 \\ \underline{0.00001000} \\ 0.00011001 \end{array}$$

Therefore, **the fuel flow rate = 0.00011001 kg/s (in binary).**

Checking in decimal: (1/15) × 1.5 = **0.10** and 0.00011001 = 0/2 + 0/4 + 0/8 + 1/16 + 1/32 + 0/64 + 0/128 + 1/256 = **0.098** (to get the closer answer of 0.10 we would need to use more significant (binary) figures.

Many times each second, a computer under the hood of an automobile receives a signal from an air flow sensor, carries out a binary computation such as the one shown in this example, and sends a signal to an actuator that causes the right amount of fuel to be injected into the air stream in order to maintain the desired air-fuel ratio. The result is much more precise and reliable control of fuel injection than was possible before computers were applied to automobiles.

9.9 BINARY CODES

We can now see how 0 and 1 can be used to represent "false" and "true" as logic values, while also of course 0 and 1 are numeric values. It is also possible to use *groups* of 0s and 1s as "codes." The now outdated Morse code is an example of a binary code, while the genetic code is based on just the pairings of four, rather than two, chemical entities known as bases.

Suppose we want to develop unique codes for the following nine basic colors: red, blue, yellow, green, black, brown, white, orange, and purple. Can we do this with a three-bit code (three 0s or 1s [bits])? No, since there are only eight combinations of three bits (note: $2^3 = 8$): namely 000, 001, 010, 011, 100, 101, 110, 111. Table 9.5 contains a possible four-bit code that would work, but of course it is only one of several, since there are $2^4 = 16$ possible combinations of a four-bit code. If we have N bits, we can code 2^N different colors.

Table 9.5 Possible set of Binary Numbers for Colors.

Color	Binary Equivalent	Color	Binary Equivalent
Red	0000	Brown	0101
Blue	0001	White	0110
Yellow	0010	Orange	0111
Green	0011	Purple	1000
Black	0100		

9.10 HOW DOES A COMPUTER WORK?

How can these abstract ideas of Boolean algebra, binary logic, and binary numbers be used to perform computations using electrical circuits, particularly switches (which we will study in a later chapter)? The modern computer is a complex device, and any answer we give you here is necessarily oversimplified. But the principles are sufficient to give you some insight. For this discussion we will need to know what a **central processing unit** (**CPU**), does and what a computer **memory** is (which can take such forms as **read-only memory** [**ROM**] and **random access memory** [**RAM**]).

The "smart" part of the computer is the CPU. It is just a series of **registers**, which is nothing but a string of switches. These switches can change their voltage states from "off" (nominally a no voltage state) to "on" or +5 V above ground. The early personal computers (or PCs) used only 8-bit registers and modern ones use 64 or 128, but we can think in terms of the 8-bit registers (The notation x, y in Table 9.6 means either state x or state y.).

Table 9.6 Eight-Bit Register.

Bit #	7	6	5	4	3	2	1	0
Voltage	0 or 5 (0,5)	0 or 5 (0,5)	0 or 5 (0,5)	0 or 5 (0,5)	0 or 5 (0,5)	0 or 5 (0,5)	0 or 5 (0,5)	0 or 5 (0,5)
Bits	0 or 1 (0,1)	0 or 1 (0,1)	0 or 1 (0,1)	0 or 1 (0,1)	0 or 1 (0,1)	0 or 1 (0,1)	0 or 1 (0,1)	0 or 1 (0,1)

This register can contain 2^8, or 256, discrete numbers or addresses. What the CPU addresses is the memory in the computer. You can think of memory as a pigeonhole bookcase with the addresses of each pigeonhole preassigned.[6]

If we have just 256 of these pigeonholes in our memory, our CPU can address each of them. If we want to calculate something, we write a computer code in a suitable language that basically says something like: Add the number in pigeonhole 37 to that in pigeonhole 64, and then put the contents in pigeonhole 134. The binary bits can represent numbers, logic statements, and so on. All we then need to do is to be able to send out the results of our binary calculations in a form we can read. For example, we can decide that the contents of pigeonhole 134 is an equivalent decimal number.

If, in the above example, we identify the program input as what is in pigeonholes 37 and 64, and the program output is pigeonhole 134, the computer is constructed essentially as in Fig. 9.3 to accomplish its mission.

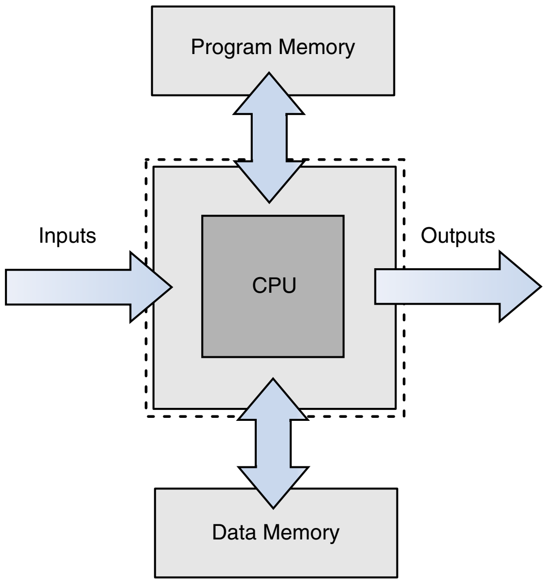


FIGURE 9.3

The central processing unit (CPU).

A computer is made up of **hardware** (electrical circuits containing such components as transistors). How the computer is instructed to execute its functions is managed by **software** (the instructions that tell the components what to do).

Part of the hardware of the computer that we have so far ignored is an internal clock. One might view the software as a list of instructions telling the computer what to do every time the clock ticks. This list of instructions includes the normal

[6]M. Sargent III and R. L. Shoemaker, *The IBM PC from the Inside Out* (Reading, MA: Addison–Wesley Publishing Co. Inc., revised edition, 1986), p. 21.

housekeeping functions that the computer carries out regularly (such as checking whether a user has entered in any keystrokes on the keyboard in the very short time interval since the last time this was checked). But it can also include downloading into a portion of the computer's memory called **program memory**, a special list of instructions called a **stored program**, and then executing that stored program. Examples of stored programs are a word processor and a spreadsheet program.

Once this stored program is downloaded into the memory, the CPU carries out this stored program step by step. Some of those steps involve carrying out computations, which are executed in binary arithmetic by the CPU using the methods described in this chapter. In some cases, the result of the computation determines which of the stored program's instructions is the next one that should be carried out. This flexibility regarding the order in which instructions are carried out is the basis of the computer's versatility as an information processing system.

Software in this binary form is called **machine language**. It is the only language that the computer understands. It is, however, a difficult language for a human to write or read. So computer engineers have developed programs that can be stored in the computer that translate software from a language that humans understand into machine language.

The types of language that humans understand are called **higher-level languages.** These languages consist of a list of statements somewhat resembling ordinary English. For example, a higher-level language might contain a statement such as "if $x<0$, then $y=36$." Examples of higher-level languages are C++, BASIC, and Java. Most computer programs are originally written in a high-level language.

The computer then **translates** this high-level language into machine language. This translation is typically carried out in two steps. First, a computer program called a **compiler** translates the statements of the high-level language into statements in a language called **assembly language** that is closer to the language that the computer understands. Then another computer program called an **assembler** translates the assembly language program into a **machine language program.** It is the machine language program that is actually executed by the computer.

The personal computers that we all use, typically by entering information through such devices as a keyboard or mouse, and producing outputs on a screen or via a printer, are called **general-purpose computers**. As the name suggests, they can be used for a wide range of purposes, from game playing to accounting to word processing to monitoring scientific apparatus.

Not all computers need this wide range of versatility. There is another important class of computers directed at narrower ranges of tasks. These computers are called **embedded computers** (Fig. 9.4).

As their name suggests, these computers are embedded within a larger system. They are not accessible by keyboard or mouse, but rather receive their inputs from sensors within that larger system. In an automobile, there may be dozens of embedded computers. There might be, for example, an embedded computer for controlling a car's stereo system, another recording data for automated service diagnostics, another for the operation of the brakes, another for the steering conditions, and yet another for fuel control.

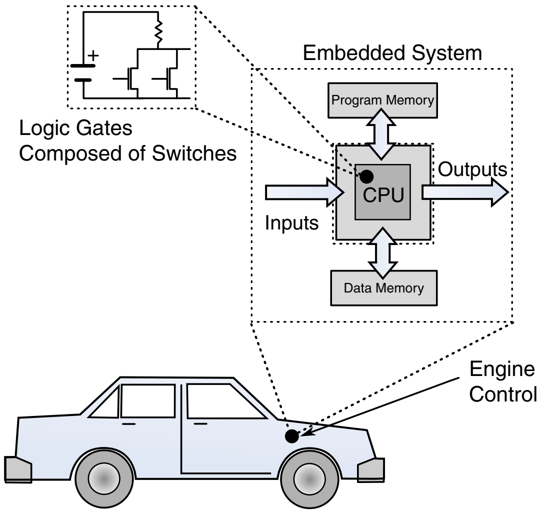


FIGURE 9.4

Schematic for an embedded automotive computer.

9.11 COMPUTER SECURITY

Computer security (or cyber security) is the protection of computer systems from theft or damage to their hardware, software or electronic data, as well as from disruption or of the services they provide. The field is growing in importance because of our increasing use of the Internet and of wireless networks such as Bluetooth and Wi-Fi, and of "smart" devices such as cellphones.

Malware is short for ***mal***icious soft***ware*** that can be used to compromise computer functions, steal data, bypass access controls, or otherwise cause harm to your computer. Malware is a broad term that refers to a variety of malicious programs. The most common types of malware are shown in Table 9.7.

These steps will help to safeguard your computer:

- Use antivirus protection and a firewall.
- Get antispyware software.
- Increase your browser security settings.
- Avoid questionable Web sites.
- Only download software from sites you trust.
- Don't open messages from unknown senders.
- Immediately delete messages you suspect to be spam.

SUMMARY

Although analog computers have been of some historical importance, digital computers do the most important work in today's advanced technologies. An engineer today must understand the principles of digital computation that rest on the immensely

Table 9.7 Common Types of Computer Malware.

Trojan Horse	A trojan horse is program that is packaged with an application, usually free, such as a screen saver or computer game. Once the program initiates, a virus is released that creates problems for your computer without your knowledge.
Virus	Computer viruses infect computers to gain control and steal data. They are spread through by visiting an infected website, clicking on an executable email file, viewing an infected advertisement, or connecting to infected removable storage devices such as USB drives.
Worm	A worm is software that copies itself repeatedly into a computer's memory using up all available RAM. When you open an email attachment containing a worm it looks through your address books choosing names at random and sends them copies of itself.
Adware	Pop-up advertisements. It is not uncommon for adware to come bundled with spyware that is capable of tracking user activity and stealing information.
Bot	A bot (short for "robot") is an automated internet program that perform functions such as capturing email addresses from website contact forms, address books, and email programs, then add them to a spam mailing list.
Spyware	Spyware is software that monitors a user's activity without their knowledge, such as collecting keystrokes and data harvesting (account information, internet logins, financial data, etc.).
Phishing	Phishing is a term used to describe individuals who try to scam users. They send emails or create web pages that look like legitimate companies designed to collect an individual's online bank, credit card, or other login information.

powerful concepts of **Boolean algebra**, **binary logic**, **truth tables**, **binary arithmetic**, and **binary codes**. These concepts enable an engineer to make a first effort at defining the concept of information. Consequently, **computer security** is one of the major computer challenges of the 21st century.

EXERCISES

1. A popular ditty of the late 19th and early 20th-century railroad era had the following words.

Passengers will please refrain From flushing toilets While the train Is standing in the station I love you.

It is sung to the tune of *Humoresque* by the 19th-century Czech composer Antonin Dvorak.[7] For this little ditty, define a variable ***S*** expressing whether or not the train is in the station, a variable ***M*** expressing whether or not the train is moving ("standing" meaning "not moving"), and a variable ***F*** expressing

[7]You can find the music at: http://www.youtube.com/watch?v=WmAZoexenx8.

the fact that the toilet may be flushed. (**Ans.: S = "the train is the station," M = "the train is moving"** (or you could use its negation, M' **meaning the train is stationary**), and **F = "the toilet may be flushed"**).

2. For the ditty in Exercise 1, **(a)** express as a logic formula the conditions under which one may flush the toilet, **(b)** evaluate the formula you wrote in (a) for $M = 1$ and $S = 1$, and **(c)** express in words the meaning of your answer to (b). (Assume that any behavior not explicitly forbidden is allowed.)
3. Rework Example 9.2 to express and evaluate the logic formula to answer the question "If the speed is below the set speed and the set speed is above the speed limit, then will the throttle be opened?"
4. In Example 9.3, include the additional Boolean variable that for actuation of the warning light the driver's door must be closed (D_{door} is true if the driver's door is closed).
5. Consider the following logic variables for a car:
   Db = "the driver's seat belt is fastened"
   Pb = "the passenger seat belt is fastened"
   W = "the seatbelt warning should be on in my car"
   Write a sentence in English that expresses the logic equation $W = Db' + Pb'$. (Ans: W = "If either the driver's seat belt is not fastened or the passenger's seat belt is not fastened, the seatbelt warning light should be on.")
6. Consider the following logic variables for a car:
   W = "the seatbelt warning light is on"
   D = "a door of the car is open"
   Ps = "there is a passenger in the passenger seat"
   K = "the key is in the ignition"
   M = "the motor is running"
   Db = "the driver's seat belt is fastened"
   Pb = "the passenger seat belt is fastened"
   Write a logic equation for W that expresses the following sentence: "If all the doors of the car are closed, and the key is in the ignition, and either the driver's seat belt is not fastened or there is a passenger in the passenger seat and the passenger's seat belt is not fastened, then the seatbelt warning light should be on."
7. Consider the following logic variables for a car:
   M = "the motor is running"
   Db = "the driver's seat belt is fastened"
   In the early 1970s the government ordered all seat belt warnings to be tied to the motor in a manner expressed by the following sentence: "If the driver's seat belt is not fastened, then the motor cannot be running."
   a. Write a logic equation for M in terms of Db that expresses this sentence.
   b. Write a truth table for that logic equation.

In practice, this seat belt light logic caused problems. Think of trying to open a manual garage door or pick up the mail from a driveway mailbox. The government soon retreated from an aroused public. **Ans.: a. $M = Db$; b. see table below.**

Db	***M***
0	0
1	1

9. Consider the sentence "If a customer at a restaurant is over 21 and shows proper identification, then she can order an alcoholic beverage." Using the following logic variables:

A = "a customer at a restaurant orders an alcoholic beverage"
I = "the customer shows proper identification"
M = "a customer at a restaurant is over 21"

a. Express this sentence as a logic equation, and
b. write a truth table for the logic equation.

T	***L***	***P***
0	0	0
0	1	1
1	0	1
1	1	1

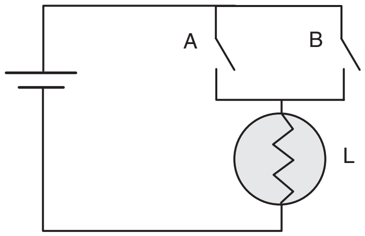
9. Consider the following variables expressing a football team's strategy.

T = "it is third down"
L = "we must gain more than 8 yards to get a first down"
P = "we will throw a pass"

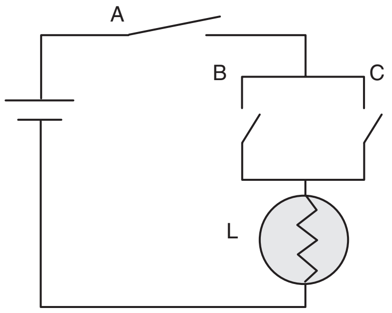
The team's strategy is expressed by this truth table. Write a logic equation for P in terms of T and L. (**Ans.:** $\boldsymbol{P = T + L}$)

Exercises 10 and 11 use electrical circuits to effect logical statements. If you are uncertain about electrical circuits, you should review ahead to the chapter on Electrical Engineering. In particular, an electrical switch (shown as an inclined line when open) means there is no current flowing from the battery. The circuit is off and is then said to be in a "0" or a "false" state; contrarily, when the switch is closed, current flows from the battery and a voltage appears across the lamp L. The circuit is now in an "on" position and is said to be "1" or a "true" state.

10. Consider the following electric circuit and the variables L = "the light is on," A = "switch A is closed," and B = "switch B is closed." Express the relationship depicted by the electric circuit as a logic equation for L in terms of A and B. (**Ans.:** $\boldsymbol{L = A + B}$.)



11. Consider the following circuit diagram and the variables L = "the light is on," A = "switch A is closed," B = "switch B is closed," and C = "switch C is closed." Write a logic equation for L in terms of A, B, and C.



12. Consider the following logic variables for a car:

1. W = "the seat belt warning light is on"
2. Ps = "there is a passenger in the passenger seat"
3. Db = "the driver's seat belt is fastened"
4. Pb = "the passenger seat belt is fastened"

Draw a circuit diagram for the logic equation $W = Db' + (Ps \cdot Pb')$.

13. Explain the following sentences.

a) "There are 10 kinds of people in the world, those who understand binary numbers, and those who don't".

b) "Binary is as easy as 1, 10, 11".

14. Convert the following numbers from binary to decimal: (a) 110, (b) 1110, and (c) 101,011. **Partial Ans.: (a) in the table below.**

–	**2^5**	**2^4**	**2^3**	**2^2**	**2^1**	**2^0**	**Decimal equivalent**
–	32	16	8	4	2	1	–
(a) 110	**0 × 32**	**0 × 16**	**0 × 8**	**1 × 4**	**1 × 2**	**0 × 1**	**6**
(b) 1110							
(c) 101011							

15. Convert the following numbers from decimal to binary: (a) 53, (b) 446, and (c) 1492. **Partial Ans: (a) in the table below.**

Decimal	1024	512	256	128	64	32	16	8	4	2	1
Binary place	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
(a) 53	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**1**
(b) 446											
(c) 1492											

16. Do the binary additions in the table below. Check your answer by converting each binary number into decimal. **Partial Ans.: the first addition in the table.**

Binary	Decimal	Binary	Decimal	Binary	Decimal
1010 +110 **10000**	**10** **6** **16**	11,101 +10,011		10,111 +10	

17. Do the binary subtractions in the table below. Check your answer by converting each binary number into decimal. **Partial Ans.: The first subtraction in the table.**

Binary	Decimal	Binary	Decimal	Binary	Decimal
1010 –110 **0100**	**10** **–6** **4**	11,101 –10,011		10,000 –1	

18. If a powerful race car has an $(A/F)_{Mass}$ of 12.0 (kg air)/(kg fuel), and the air intake draws in 1.000 kg of air per second, how much fuel must be injected every second? Solve in binary to three significant binary digits**.** (**Ans. 0.000,101 kg.**)

19. Suppose we want to devise a binary code to represent the fuel levels in a car:
- **a.** If we need only to describe the possible levels (empty, 1/4 full, 1/2 full, 3/4 full, and full), how many bits are needed?
- **b.** Give one possible binary code that describes the levels in (a).
- **c.** If we need to describe the levels (empty, 1/8 full, 1/4 full, 3/8 full, 1/2 full, 5/8 full, 3/4 full, 7/8 full, and full), how many bits would be needed?
- **d.** If we used an 8-bit code, how many levels could we represent?

20. Construct a spreadsheet[8] that converts binary numbers from 0 to 111 to decimal numbers, print as formulae using the "control tilde" command. Check your spreadsheet against Exercise14. (**Ans. e.g., binary 110 ≡ $1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$ = decimal 6.**)

21. Construct a spreadsheet that converts decimal number 53 to binary. Print as formulae using the "control tilde" command. Check your spreadsheet against Exercise 15. **Hint:** 5 (decimal) can be divided by 2^2 to yield an integer "1" and remainder 1; 1 can't be divided by 2^1 [therefore, integer "0"]; and "1" can be divided by 2^0 for the last integer "1." Check for a spreadsheet function that will divide two numbers and display their result with no remainder.

22. Construct a spreadsheet that does binary subtraction with one's complement. Test it on Exercise 17.

[8]Spreadsheets have some built-in functions for decimal conversions to and from binary. It is recommended that you try first to use the actual mathematical functions described in this chapter and then *check* your answers using these functions to confirm those answers.

23. In the game "Rock, Scissors, Paper" we have the following rules:

Rock breaks scissors
Scissors cuts paper
Paper covers rock

If we were to create a code to represent the three entities, Rock, Scissors, and Paper, we would need two bits. Suppose we have the following code, where we call the first bit X and the second bit Y:

	X	Y
Rock:	0	0
Scissors:	0	1
Paper:	1	0

Now if we have two players who can each choose one of these codes, we can play the game. For example,

Player 1	Rock (0,0)	Player 2	Scissors (0,1)	Player 1 wins
Player 1	Scissors (0,1)	Player 2	Scissors (0,1)	Tie
Player 1	Rock (0,0)	Player 2	Paper (1,0)	Player 2 wins

We see that there are three possible outcomes of the game: Player 1 wins, Tie, Player 2 wins. Complete the table below that describes all the possible outcomes of the game.

Player 1 X,Y	**Player 2 X,Y**	**Player 1 Wins**	**Tie**	**Player 2 Wins**
0,0	0,0	0	1	0
0,0	0,1	1	0	0
0,0	1,0			
0,1	0,0			
0,1	0,1			
0,1	1,0			
1,0	0,0			
1,0	0,1			
1,0	1,0			
Totals				

24. A company purchased a computer program for your part-time job with them. The license agreement states that you can make a backup copy, but you can only use the program on one computer at a time. Since you have permission to make a backup copy, why not make copies for friends? What do you do? (Use the Engineering Ethics Matrix.)

a. Go ahead, since your friends only use one computer at a time, and these are backup copies.
b. Make the backup copy but sharing it with anyone clearly violates the license agreement.
c. Ask your supervisor if you can use the backup copy at home, and then make as many copies as you wish.
d. Use the program discretely, since software license agreements can't be enforced anyway.

25. You are a software engineer at a small company. You have written a software program that will be used by a major manufacturer in a popular product line. Your supervisor asks you to install a "back door" into the program that no one will know about so that he can monitor its use by the public. What do you do? (Use the Engineering Ethics Matrix.)
    a. Install the back door, since it sounds like a fun experiment.
    b. Tell your supervisor that you can't do it without authorization from the end user.
    c. Install the back door but then deactivate it before the software is implemented.
    d. Stall your supervisor while you look for another job.