human beings rather than the laws of physics. A traffic light's behavior is predominantly defined by human beings rather than by natural physical laws. This book is concerned with the design of digital systems that are suited to the algorithmic requirements of their particular range of applications. Digital logic and arithmetic are critical building blocks in constructing such systems.

An algorithm is a procedure for solving a problem through a series of finite and specific steps. It can be represented as a set of mathematical formulas, lists of sequential operations, or any combination thereof. Each of these finite steps can be represented by a *Boolean logic* equation. Boolean logic is a branch of mathematics that was discovered in the nineteenth century by an English mathematician named George Boole. The basic theory is that logical relationships can be modeled by algebraic equations. Rather than using arithmetic operations such as addition and subtraction, Boolean algebra employs logical operations including AND, OR, and NOT. Boolean variables have two enumerated values: true and false, represented numerically as 1 and 0, respectively.

The AND operation is mathematically defined as the product of two Boolean values, denoted A and B for reference. *Truth tables* are often used to illustrate logical relationships as shown for the AND operation in Table 1.1. A truth table provides a direct mapping between the possible inputs and outputs. A basic AND operation has two inputs with four possible combinations, because each input can be either 1 or 0 — true or false. Mathematical rules apply to Boolean algebra, resulting in a non-zero product only when both inputs are 1.

TABLE 1.1 AND Operation Truth Table

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

Summation is represented by the OR operation in Boolean algebra as shown in Table 1.2. Only one combination of inputs to the OR operation result in a zero sum: $0 + 0 = 0$.

TABLE 1.2 OR Operation Truth Table

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

AND and OR are referred to as *binary operators,* because they require two operands. NOT is a *unary operator*, meaning that it requires only one operand. The NOT operator returns the complement of the input: 1 becomes 0, and 0 becomes 1. When a variable is passed through a NOT operator, it is said to be *inverted*.

Boolean variables may not seem too interesting on their own. It is what they can be made to represent that leads to useful constructs. A rather contrived example can be made from the following logical statement:

"If today is Saturday or Sunday and it is warm, then put on shorts."

Three Boolean inputs can be inferred from this statement: Saturday, Sunday, and warm. One Boolean output can be inferred: shorts. These four variables can be assembled into a single logic equation that computes the desired result,

shorts = (Saturday OR Sunday) AND warm

While this is a simple example, it is representative of the fact that any logical relationship can be expressed algebraically with products and sums by combining the basic logic functions AND, OR, and NOT.

Several other logic functions are regarded as elemental, even though they can be broken down into AND, OR, and NOT functions. These are not–AND (NAND), not–OR (NOR), exclusive–OR (XOR), and exclusive–NOR (XNOR). Table 1.3 presents the logical definitions of these other basic functions. XOR is an interesting function, because it implements a sum that is distinct from OR by taking into account that 1 + 1 does not equal 1. As will be seen later, XOR plays a key role in arithmetic for this reason.

TABLE 1.3 NAND, NOR, XOR, XNOR Truth Table

A	B	A NAND B	A NOR B	A XOR B	A XNOR B
0	0	1	1	0	1
0	1	1	0	1	0
1	0	1	0	1	0
1	1	0	0	0	1

All binary operators can be chained together to implement a wide function of any number of inputs. For example, the truth table for a ten-input AND function would result in a 1 output only when all inputs are 1. Similarly, the truth table for a seven-input OR function would result in a 1 output if any of the seven inputs are 1. A four-input XOR, however, will only result in a 1 output if there are an odd number of ones at the inputs. This is because of the logical daisy chaining of multiple binary XOR operations. As shown in Table 1.3, an even number of 1s presented to an XOR function cancel each other out.

It quickly grows unwieldy to write out the names of logical operators. Concise algebraic expressions are written by using the graphical representations shown in Table 1.4. Note that each operation has multiple symbolic representations. The choice of representation is a matter of style when handwritten and is predetermined when programming a computer by the syntactical requirements of each computer programming language.

A common means of representing the output of a generic logical function is with the variable Y. Therefore, the AND function of two variables, A and B, can be written as Y = A & B or Y = A*B. As with normal mathematical notation, products can also be written by placing terms right next to each other, such as Y = AB. Notation for the inverted functions, NAND, NOR, and XNOR, is achieved by