is irrelevant: $\overline{A}C$. It may appear tempting to create a product term consisting of the three boxes on the bottom edge of the K-map. This is not valid because it does not result in all boxes sharing a common product relationship, and therefore violates the power-of-two rule mentioned previously. Upon completing the K-map, all product terms are summed to yield a final and simplified Boolean equation that relates the input variables and the output: $Y = \overline{B} + \overline{A}C$.

Functions of four variables are just as easy to solve using a K-map. Beyond four variables, it is preferable to break complex functions into smaller subfunctions and then combine the Boolean equations once they have been determined. Figure 1.6 shows an example of a completed Karnaugh map for a hypothetical function of four variables. Note the overlap between several groups to achieve a simplified set of product terms. The lager a group is, the fewer unique terms will be required to represent its logic. There is nothing to lose and something to gain by forming a larger group whenever possible. This K-map has four product terms that are summed for a final result: $Y = \overline{A}\ \overline{C} + \overline{B}\ \overline{C} + \overline{A}B\overline{D} + ABCD$.

In both preceding examples, each result box in the truth table and Karnaugh map had a clearly defined state. Some logical relationships, however, do not require that every possible result necessarily be a one or a zero. For example, out of 16 possible results from the combination of four variables, only 14 results may be mandated by the application. This may sound odd, but one explanation could be that the particular application simply cannot provide the full 16 combinations of inputs. The specific reasons for this are as numerous as the many different applications that exist. In such circumstances these so-called *don't care* results can be used to reduce the complexity of your logic. Because the application does not care what result is generated for these few combinations, you can arbitrarily set the results to 0s or 1s so that the logic is minimized. Figure 1.7 is an example that modifies the Karnaugh map in Fig. 1.6 such that two don't care boxes are present. Don't care values are most commonly represented with "x" characters. The presence of one x enables simplification of the resulting logic by converting it to a 1 and grouping it with an adjacent 1. The other x is set to 0 so that it does not waste additional logic terms. The new Boolean equation is simplified by removing B from the last term, yielding $Y = \overline{A}\ \overline{C} + \overline{B}\ \overline{C} + \overline{A}B\overline{D} + ACD$. It is helpful to remember that x values can generally work to your benefit, because their presence imposes fewer requirements on the logic that you must create to get the job done.

1.4 BINARY AND HEXADECIMAL NUMBERING

The fact that there are only two valid Boolean values, 1 and 0, makes the *binary* numbering system appropriate for logical expression and, therefore, for digital systems. Binary is a base-2 system in

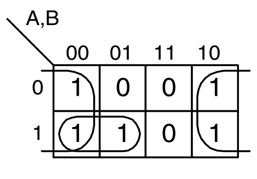

FIGURE 1.5 Completed Karnaugh map for a function of three variables.

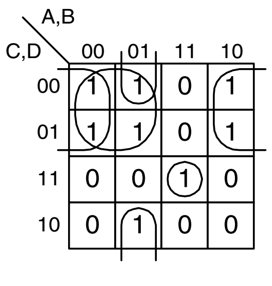

FIGURE 1.6 Completed Karnaugh map for function of four variables.

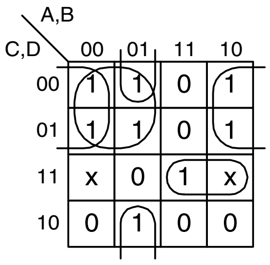

FIGURE 1.7 Karnaugh map for function of four variables with two "don't care" values.

which only the digits 1 and 0 exist. Binary follows the same laws of mathematics as decimal, or base-10, numbering. In decimal, the number 191 is understood to mean one hundreds plus nine tens plus one ones. It has this meaning, because each digit represents a successively higher power of ten as it moves farther left of the decimal point. Representing 191 in mathematical terms to illustrate these increasing powers of ten can be done as follows:

$$191 = 1 \times 10^2 + 9 \times 10^1 + 1 \times 10^0$$

Binary follows the same rule, but instead of powers of ten, it works on powers of two. The number 110 in binary (written as 110_2 to explicitly denote base 2) does not equal 110_{10} (decimal). Rather, $110_2 = 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 6_{10}$. The number 191_{10} can be converted to binary by performing successive division by decreasing powers of 2 as shown below:

$$\begin{array}{lll}
191 \div 2^7 & = 191 \div 128 & = 1 \text{ remainder } 63 \\
63 \div 2^6 & = 63 \div 64 & = 0 \text{ remainder } 63 \\
63 \div 2^5 & = 63 \div 32 & = 1 \text{ remainder } 31 \\
31 \div 2^4 & = 31 \div 16 & = 1 \text{ remainder } 15 \\
15 \div 2^3 & = 15 \div 8 & = 1 \text{ remainder } 7 \\
7 \div 2^2 & = 7 \div 4 & = 1 \text{ remainder } 3 \\
3 \div 2^1 & = 3 \div 2 & = 1 \text{ remainder } 1 \\
1 \div 2^0 & = 1 \div 1 & = 1 \text{ remainder } 0
\end{array}$$

The final result is that $191_{10} = 10111111_2$. Each binary digit is referred to as a *bit*. A group of N bits can represent decimal numbers from 0 to $2^N - 1$. There are eight bits in a *byte*, more formally called an *octet* in certain circles, enabling a byte to represent numbers up to $2^8 - 1 = 255$. The preceding example shows the eight power-of-two terms in a byte. If each term, or bit, has its maximum value of 1, the result is 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255.

While binary notation directly represents digital logic states, it is rather cumbersome to work with, because one quickly ends up with long strings of ones and zeroes. *Hexadecimal*, or base 16 (*hex* for short), is a convenient means of representing binary numbers in a more succinct notation. Hex matches up very well with binary, because one hex digit represents four binary digits, given that