6

COMBINATIONAL LOGIC CIRCUITS

In the previous chapter, you learned about a computer's basic component, the logic gate. Computers are constructed from assemblages of logic gates, called *logic circuits*, that process digital information. In this and the following two chapters, we'll look at how to build some of the logic circuits that make up CPUs, memory, and other devices. We won't describe any of these units in their entirety; instead, we'll look at a few small parts and discuss the concepts behind them. The goal is to provide an introductory overview of the ideas that underlie these logic circuits.

The Two Classes of Logic Circuits

Logic circuits come in two classes:

Combinational
A *combinational logic circuit* has output(s) that depend only on the input(s)s given at any specific time, and not on any previous input(s)s.

Sequential
A *sequential logic circuit* has output(s)s that depend both on previous and current input(s)s.

To elucidate these two types, let's consider a TV remote. You can select a specific channel by entering a number on the remote. The channel selection depends only on the number you entered, and ignores the channels you were viewing before. Thus, the relationship between the input and the output is combinational.

Formatted: RunInHead

Half Adder

Addition can be done with several kinds of circuits. We'll start with the *half adder*, which simply adds the two bits in the current bit position of a number (expressed in binary). This is shown by the truth table, Table 6-1. In this table, *xi* is the *ith* bit of the number *x*. The values in the *yi* column represent the *ith* bit of the number *y*. *Sumi* is the *ith* bit of the number, *Sum*, and *Carryi+1* is the carry from adding bits *xi* and *yi*.

Table 6-1 Adding Two Bits, Half-Adder

xi	*yi*	*Carryi+1*	*Sumi*
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

The sum is the XOR of the two inputs, and the carry is the AND of the two inputs. Figure 6-1 shows ~~The~~ the logic circuit for a half adder ~~is shown in Figure 6-1~~.

INCLUDE X86-64_06-01.SVG HERE.

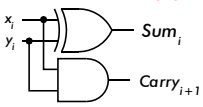

Figure 6-1 ~~Figure 6-1:~~ A half adder circuit

But there's a flaw here: the half adder works with only two input bits. It can be used to add the two bits from the same bit position of two numbers, but it doesn't take into account a possible carry from the next lower-order bit position. To allow for the carry, we'll have to add a third input.

Full Adder

Unlike the half adder, a *full adder* circuit has three one-bit inputs, *Carryi*, *xi*, and *yi*. *Carryi* is the carry that resulted when you added the two bits in the previous bit position (the bit to the right). For example, if we're adding the two bits in bit position 5, the inputs to the full adder are the two bits in position 5 plus the carry from adding the bits in bit position 4. Table 6-2 shows ~~The~~ the results ~~are shown in Table 6-2~~.

Table 6-2 ~~Table 6-2:~~ Adding Two Bits, Full Adder

Carryi	*xi*	*yi*	*Carryi+1*	*Sumi*
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

To design a full adder circuit, we start with the function that specifies when *Sumi* is 1 as a sum of

Formatted: Italic
Formatted: Italic, Not Superscript/ Subscript
Formatted: Italic
Formatted: Italic, Not Superscript/ Subscript
Formatted: Italic
Formatted: Italic, Font: 9 pt
Formatted: Italic
Formatted: Italic, Not Superscript/ Subscript
Formatted: Italic
Formatted: Italic, Font: (Default) +Body (Calibri), Not Italic
Formatted: Italic, Font: (Default) +Body (Calibri), Not Italic, Not Superscript/ Subscript
Formatted: Italic
Formatted: Italic, Font: (Default) +Body (Calibri), Not Italic
Formatted: Italic, Font: (Default) +Body (Calibri), Not Italic, Not Superscript/ Subscript
Formatted: Italic
Formatted: Italic, Font: (Default) +Body (Calibri), Not Italic
Formatted: Italic, Font: (Default) +Body (Calibri), Not Italic, Not Superscript/ Subscript
Formatted: Italic
Formatted: Italic, Font: (Default) +Body (Calibri), Not Italic
Formatted: Italic, Font: (Default) +Body (Calibri), 11 pt, Not Italic
Formatted: Italic, Font: (Default) +Body (Calibri), 11 pt
Formatted: Italic
Formatted: Italic, Font: (Default) +Body (Calibri), Not Italic
Formatted: Italic, Font: (Default) +Body (Calibri), Not Italic, Not Superscript/ Subscript
Formatted: Italic

product terms from Table 6-2.

$$Sum_i(Carry_i, x_i, y_i) = (\neg Carry_i \wedge \neg x_i \wedge y_i) \vee (\neg Carry_i \wedge x_i \wedge \neg y_i) \vee (Carry_i \wedge \neg x_i \wedge \neg y_i) \vee (Carry_i \wedge x_i \wedge y_i)$$

There are no obvious simplifications in this equation, so let's look at the Karnaugh map for *Sumi* (Figure 6-2).

INSERT x86-64_06-02.SVG HERE.

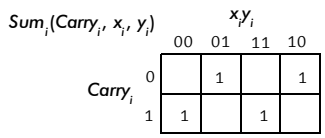

Figure 6-2 A Karnaugh map for sum of three bits, *Carryi*, *xi*, and *yi*

There are no obvious groupings in Figure 6-2, so we are left with the four product terms to compute *Sumi* in the previous equation.

We saw in Chapter 4 that *Carryi+1* can be expressed by this equation:

$$Carry_{i+1} = (x_i \wedge y_i) \vee (Carry_i \wedge x_i) \vee (Carry_i \wedge y_i)$$

Together, these two functions give the circuit for a full adder in Figure 6-3.

INSERT x86-64_06-03.SVG HERE.

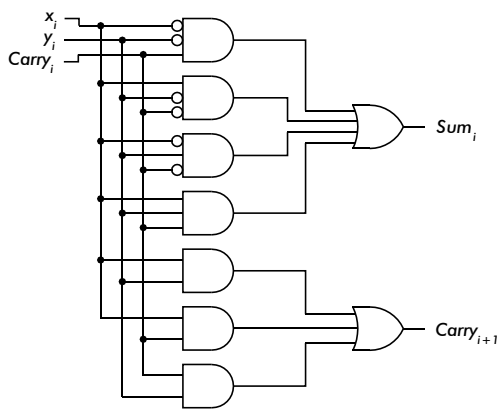

Figure 6-3 A full adder circuit

As you can see, the full adder uses nine logic gates. In the next section, we'll see if we can find a simpler circuit.