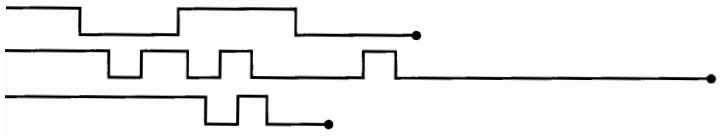
First we will see how the various arithmetic operations are performed on binary numbers using "pencil and paper," and then we will study the actual logic circuits that perform these operations in a digital system.



6-1 BINARY ADDITION

The addition of two binary numbers is performed in exactly the same manner as the addition of decimal numbers. In fact, binary addition is simpler, since there are fewer cases to learn. Let us first review decimal addition:

$$\begin{array}{rrrl} & & \downarrow & \downarrow \\ 3 & 7 & 6 & \text{LSD} \\ +4 & 6 & 1 & \\ \hline 8 & 3 & 7 & \end{array}$$

The least-significant-digit (LSD) position is operated on first, producing a sum of 7. The digits in the second position are then added to produce a sum of 13, which produces a **carry** of 1 into the third position. This produces a sum of 8 in the third position.

The same general steps are followed in binary addition. However, only four cases can occur in adding the two binary digits (bits) in any position. They are:

$$\begin{aligned} 0 + 0 &= 0 \\ 1 + 0 &= 1 \\ 1 + 1 &= 10 = 0 + \text{carry of 1 into next position} \\ 1 + 1 + 1 &= 11 = 1 + \text{carry of 1 into next position} \end{aligned}$$

The last case occurs when the two bits in a certain position are 1 and there is a carry from the previous position. Here are several examples of the addition of two binary numbers (decimal equivalents are in parentheses):

$$\begin{array}{rrr} 011\ (3) & 1001\ (9) & 11.011\ (3.375) \\ +\ 110\ (6) & +\ 1111\ (15) & +\ 10.110\ (2.750) \\ \hline 1001\ (9) & 11000\ (24) & 110.001\ (6.125) \end{array}$$

It is not necessary to consider the addition of more than two binary numbers at a time, because in all digital systems the circuitry that actually performs the addition can handle only two numbers at a time. When more than two numbers are to be added, the first two are added together and then their sum is added to the third number, and so on. This is not a serious drawback, since modern digital computers can typically perform an addition operation in several nanoseconds.

Addition is the most important arithmetic operation in digital systems. As we shall see, the operations of subtraction, multiplication, and division as they are per-

formed in most modern digital computers and calculators actually use only addition as their basic operation.

Review Question

1. Add the following pairs of binary numbers.
   (a) 10110 + 00111 (b) 011.101 + 010.010 (c) 10001111 + 00000001

6-2 REPRESENTING SIGNED NUMBERS

In digital computers, the binary numbers are represented by a set of binary storage devices (e.g., flip-flops). Each device represents one bit. For example, a six-bit FF register could store binary numbers ranging from 000000 to 111111 (0 to 63 in decimal). This represents the *magnitude* of the number. Since most digital computers and calculators handle negative as well as positive numbers, some means is required for representing the *sign* of the number (+ or −). This is usually done by adding to the number another bit called the **sign bit.** In general, the common convention which has been adopted is that a 0 in the sign bit represents a positive number and a 1 in the sign bit represents a negative number. This is illustrated in Figure 6-1. Register A contains the bits 0110100. The 0 in the leftmost bit (A_6) is the sign bit that represents +. The other six bits are the magnitude of the number 110100_2, which is equal to 52 in decimal. Thus, the number stored in the A register is +52. Similarly, the number stored in the B register is −52, since the sign bit is 1, representing −.

The sign bit is used to indicate the positive or negative nature of the stored binary number. The numbers in Figure 6-1 consist of a sign bit and six magnitude bits. The magnitude bits are the true binary equivalent of the decimal value being represented. This is called the **sign-magnitude system** for representing signed binary numbers.

Although the sign-magnitude system is straightforward, calculators and computers do not normally use it, because the circuit implementation is more complex than in other systems. The most commonly used system for representing signed binary numbers is the **2's-complement system.** Before we see how this is done, we must first see how to form the 1's complement and 2's complement of a binary number.

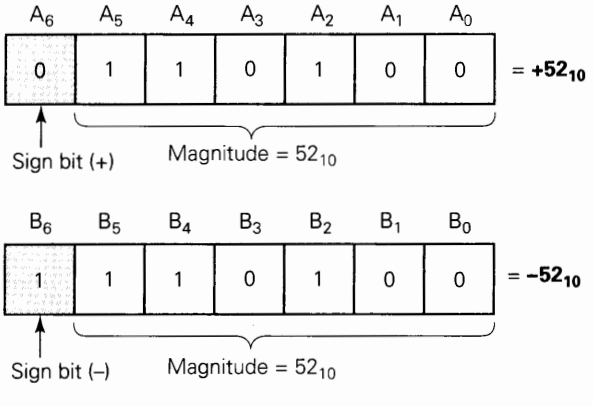


FIGURE 6-1 Representation of signed numbers in sign-magnitude form.

1's-Complement Form

The 1's complement of a binary number is obtained by changing each 0 to a 1 and each 1 to a 0. In other words, change each bit in the number to its complement. The process is shown below.

```
1 0 1 1 0 1    original binary number
↓ ↓ ↓ ↓ ↓ ↓
0 1 0 0 1 0    complement each bit to form 1's complement
```

Thus, we say that the 1's complement of 101101 is 010010.

2's Complement Form

The 2's complement of a binary number is formed by taking the 1's complement of the number and adding 1 to the least-significant-bit position. The process is illustrated below for $101101_2 = 45_{10}$.

```
  1 0 1 1 0 1    binary equivalent of 45
  0 1 0 0 1 0    complement each bit to form 1's complement
+         1      add 1 to form 2's complement
  0 1 0 0 1 1    2's complement of original binary number
```

Thus, we say that 010011 is the 2's complement representation of 101101.

Here's another example of converting a binary number to its 2's-complement representation:

```
  1 0 1 1 0 0    original binary number
  0 1 0 0 1 1    1's complement
+         1      add 1
  0 1 0 1 0 0    2's complement of original number
```

Representing Signed Numbers Using 2's Complement

The 2's-complement system for representing signed numbers works like this:

- If the number is positive, the magnitude is represented in its true binary form, and a sign bit of 0 is placed in front of the MSB. This is shown in Figure 6-2 for $+45_{10}$.

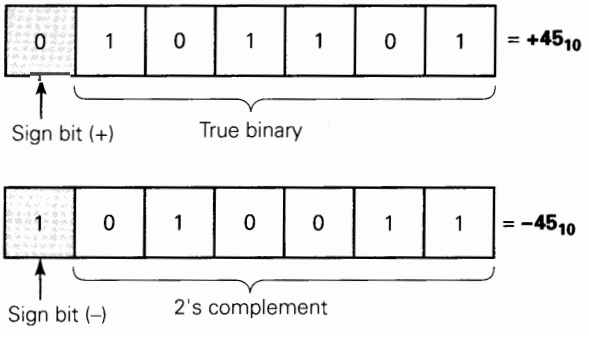


FIGURE 6-2 Representation of signed numbers in the 2's-complement system.

- If the number is negative, the magnitude is represented in its 2's-complement form, and a sign bit of 1 is placed in front of the MSB. This is shown in Figure 6-2 for -45_{10}.

The 2's-complement system is used to represent signed numbers because, as we shall see, it allows us to perform the operation of subtraction by actually performing addition. This is significant because it means that a digital computer can use the same circuitry both to add and subtract, thereby realizing a saving in hardware.

EXAMPLE 6-1

Represent each of the following signed decimal numbers as a signed binary number in the 2's-complement system. Use a total of five bits including the sign bit.

(a) +13 (b) −9 (c) +3 (d) −2 (e) −8

Solution

(a) Since the number is positive, the magnitude (13) will be represented in its true-magnitude form, that is, $13 = 1101_2$. Attaching the sign bit of 0, we have

$$+13 = 01101$$

sign bit ↑

(b) Since the number is negative, the magnitude (9) must be represented in 2's-complement form:

$$\begin{array}{rll} 9_{10} = 1001_2 & & \\ \quad 0110 & \text{(1's complement)} \\ +\quad 1 & \text{(add 1 to LSB)} \\ \quad 0111 & \text{(2's complement)} \end{array}$$

When we attach the sign bit of 1, the complete signed number becomes

$$-9 = 10111$$

sign bit ↑

The procedure we have just followed required two steps. First we determined the 2's complement of the magnitude, and then we attached the sign bit. This can be accomplished in one step if we include the sign bit in the 2's-complement process. For example, to find the representation for −9, we start with the representation for +9, *including the sign bit,* and we take the 2's complement of it in order to obtain the representation for −9.

$$\begin{array}{rll} +9 = 01001 & & \\ \quad 10110 & \text{(1's complement of each bit including sign bit)} \\ +\quad 1 & \text{(add 1 to LSB)} \\ -9 = 10111 & \text{(2's-complement representation of } -9\text{)} \end{array}$$

The result is, of course, the same as before.

(c) The decimal value 3 can be represented in binary using only two bits. However, the problem statement requires a four-bit magnitude preceded by a sign bit. Thus, we have

$$+3_{10} = 00011$$

In many situations the number of bits is fixed by the size of the registers that will be holding the binary numbers, so that 0s may have to be added in order to fill the required number of bit positions.

(d) Start by writing +2 using five bits:

+2 =	00010	
	11101	(1's complement)
+	1	(add 1)
−2 =	**11110**	(2's-complement representation of −2)

(e) Start with +8:

+8 =	01000	
	10111	(complement each bit)
+	1	(add 1)
−8 =	**11000**	(2's-complement representation of −8)

Negation

Negation is the operation of converting a positive number to its negative equivalent or a negative number to its positive equivalent. When signed binary numbers are represented in the 2's-complement system, negation is performed simply by performing the 2's-complement operation. To illustrate, let's start with +9. Its signed representation is 01001. If we 2's-complement this, we get 10111. Clearly, this is a negative number since the sign bit is a 1. Actually, 10111 represents −9, which is the negative equivalent of the number we started with. Likewise, we can start with the representation for −9, which is 10111. If we 2's-complement this, we get 01001, which we recognize as +9. These steps are diagrammed below.

start with → 01001 = +9
2's-complement (negate) → 10111 = −9
negate again → 01001 = +9

Thus, we negate a signed binary number by 2's-complementing it.

This negation changes the number to its equivalent of opposite sign. We used negation in steps (d) and (e) of Example 6-1 to convert positive numbers to their negative equivalents.

EXAMPLE 6-2

Each of the following numbers is a signed binary number in the 2's-complement system. Determine the decimal value in each case:
(a) 01100 (b) 11010 (c) 10001

Solution

(a) The sign bit is 0, so the number is *positive* and the other four bits represent the true magnitude of the number. That is, $1100_2 = 12_{10}$. Thus, the decimal number is **+12**.

(b) The sign bit of 11010 is a 1, so we know that the number is negative, but we can't tell what the magnitude is. We can find the magnitude by negating (2's-complementing) the number to convert it to its positive equivalent.

	11010	(original negative number)
	00101	(1's complement)
+	1	(add 1)
	00110	(+6)

Since the result of the negation is 00110 = +6, the original number 11010 must be equivalent to **−6**.

(c) Follow the same procedure as in (b):

	10001	(original negative number)
	01110	(1's complement)
+	1	(add 1)
	01111	(+15)

Thus, 10001 = **−15**.

Special Case in 2's-Complement Representation

Whenever a signed number has a 1 in the sign bit and all 0s for the magnitude bits, its decimal equivalent is -2^N, where N is the number of bits in the *magnitude*. For example,

$$1000 = -2^3 = -8$$
$$10000 = -2^4 = -16$$
$$100000 = -2^5 = -32$$

and so on.

Thus, we can state that the complete range of values that can be represented in the 2's-complement system having N magnitude bits is

$$-2^N \text{ to } +(2^N - 1)$$

There are a total of 2^{N+1} different values, *including* zero.

For example, Table 6-1 lists all signed numbers that can be represented in four bits using the 2's-complement system (note there are three magnitude bits, so $N = 3$). Note that the sequence starts at $-2^N = -2^3 = -8_{10} = 1000_2$ and proceeds upwards to $+(2^N - 1) = +2^3 - 1 = +7_{10} = 0111_2$ by adding 0001 at each step as in an up counter.

TABLE 6-1

Decimal Value	Signed Binary Using 2's Complement
$+7 = 2^3 - 1$	0111
+6	0110
+5	0101
+4	0100
+3	0011
+2	0010
+1	0001
0	0000
−1	1111
−2	1110
−3	1101
−4	1100
−5	1011
−6	1010
−7	1001
$-8 = -2^3$	1000

EXAMPLE 6-3

What is the range of *unsigned* decimal values that can be represented in a byte?

Solution

Recall that a byte is eight bits. Since we are interested in unsigned numbers here, there is no sign bit, so all of the eight bits are used for the magnitude. Therefore, the values will range from

$$00000000_2 = 0_{10}$$

to

$$11111111_2 = 255_{10}$$

This is a total of 256 different values, which we could have predicted since $2^8 = 256$.

EXAMPLE 6-4

What is the range of *signed* decimal values that can be represented in a byte?

Solution

Since the MSB is to be used as the sign bit, there are seven bits for the magnitude. The largest negative value is

$$10000000_2 = -2^7 = -128_{10}$$

The largest positive value is

$$01111111_2 = +2^7 - 1 = +127_{10}$$

Thus, the range is -128 to $+127$; this is a total of 256 different values, including zero. Alternatively, since there are seven magnitude bits ($N = 7$), then there are $2^{N+1} = 2^8 = 256$ different values.

EXAMPLE 6-5

A certain computer is storing the following two signed numbers in its memory using the 2's-complement system:

$$00011111_2 = +31_{10}$$
$$11110100_2 = -12_{10}$$

While executing a program, the computer is instructed to convert each number to its opposite sign; that is, change the $+31$ to -31 and change the -12 to $+12$. How will it do this?

Solution

This is simply the negation operation whereby a signed number can have its polarity changed simply by performing the 2's-complement operation on the *complete* number, including the sign bit. The computer circuitry will take the signed number from memory, find its 2's complement, and put the result back in memory.

Review Questions

1. Represent each of the following values as an eight-bit signed number in the 2's-complement system.
   (a) $+13$ (b) -7 (c) -128
2. Each of the following is a signed binary number in the 2's-complement system. Determine the decimal equivalent for each.
   (a) 100011 (b) 1000000 (c) 01111110
3. What range of signed decimal values can be represented in 12 bits (including the sign bit)?
4. How many bits are required to represent decimal values ranging from -50 to $+50$?
5. What is the largest negative decimal value that can be represented by a two-byte number?
6. Perform the 2's-complement operation on each of the following.
   (a) 10000 (b) 10000000 (c) 1000
7. Define the negation operation.

6-3 ADDITION IN THE 2's-COMPLEMENT SYSTEM

We will now investigate how the operations of addition and subtraction are performed in digital machines that use the 2's-complement representation for negative numbers. In the various cases to be considered, it is important to note that the sign bit of each number is operated on in the same manner as the magnitude bits.

Case I: Two Positive Numbers. The addition of two positive numbers is straightforward. Consider the addition of +9 and +4:

```
+9 →  0  1001   (augend)
+4 →  0  0100   (addend)
      0  1101   (sum = +13)
      ↑
      sign bits
```

Note that the sign bits of the **augend** and the **addend** are both 0 and the sign bit of the sum is 0, indicating that the sum is positive. Also note that the augend and the addend are made to have the same number of bits. This must *always* be done in the 2's-complement system.

Case II: Positive Number and Smaller Negative Number. Consider the addition of +9 and −4. Remember that the −4 will be in its 2's-complement form. Thus, +4 (00100) must be converted to −4 (11100).

```
          sign bits
          ↓
+9 →  0  1001   (augend)
−4 →  1  1100   (addend)
   1  0  0101
   ↑
   This carry is disregarded; the result is 00101 (sum = +5).
```

In this case, the sign bit of the addend is 1. Note that the sign bits also participate in the addition process. In fact, a carry is generated in the last position of addition. *This carry is always disregarded,* so that the final sum is 00101, which is equivalent to +5.

Case III: Positive Number and Larger Negative Number. Consider the addition of −9 and +4:

```
−9 →  10111
+4 →  00100
      11011   (sum = −5)
      ↑
      negative sign bit
```

The sum here has a sign bit of 1, indicating a negative number. Since the sum is negative, it is in 2's-complement form, so that the last four bits, 1011, actually represent the 2's complement of the sum. To find the true magnitude of the sum, we must negate (2's-complement) 11011; the result is 00101 = +5. Thus, 11011 represents −5.

Case IV: Two Negative Numbers

$$\begin{array}{rr} -9 \rightarrow & 10111 \\ -4 \rightarrow & 11100 \\ \hline & \not{1}\ 10011 \end{array}$$

↑ ↑ sign bit

↑ This carry is disregarded; the result is 10011 (sum = −13).

This final result is again negative and in 2's-complement form with a sign bit of 1. Negating (2's-complementing) this result produces 01101 = +13.

Case V: Equal and Opposite Numbers

$$\begin{array}{rr} -9 \rightarrow & 10111 \\ +9 \rightarrow & 01001 \\ \hline 0 & \not{1}\ 00000 \end{array}$$

↑ Disregard; the result is 00000 (sum = +0).

The result is obviously +0, as expected.

Review Questions

Assume the 2's-complement system for both questions.

1. *True or false:* Whenever the sum of two signed binary numbers has a sign bit of 1, the magnitude of the sum is in 2's-complement form.
2. Add the following pairs of signed numbers. Express the sum as a signed binary number and as a decimal number.
   (a) 100111 + 111011 (b) 100111 + 011001

6-4 SUBTRACTION IN THE 2's-COMPLEMENT SYSTEM

The subtraction operation using the 2's-complement system actually involves the operation of addition and is really no different from the various cases for addition considered in Section 6-3. When subtracting one binary number (the **subtrahend**) from another binary number (the **minuend**), use the following procedure:

1. **Negate the subtrahend.** This will change the subtrahend to its equivalent value of opposite sign.
2. **Add this to the minuend.** The result of this addition will represent the *difference* between the subtrahend and the minuend.

Once again, as in all 2's-complement arithmetic operations, it is necessary that both numbers have the same number of bits in their representations.

Let us consider the case where +4 is to be subtracted from +9.

$$\begin{array}{rr} \text{minuend } (+9) \rightarrow & 01001 \\ \text{subtrahend } (+4) \rightarrow & 00100 \end{array}$$

Negate the subtrahend to produce 11100, which represents −4. Now add this to the minuend.

```
  01001    (+9)
+ 11100    (−4)
  00101    (+5)
↑
└ Disregard, so the result is 00101 = +5.
```

When the subtrahend is changed to its 2's complement, it actually becomes −4, so that we are *adding* −4 and +9, which is the same as subtracting +4 from +9. This is the same as case II of Section 6-3. Any subtraction operation, then, actually becomes one of addition when the 2's-complement system is used. This feature of the 2's-complement system has made it the most widely used of the methods available, since it allows addition and subtraction to be performed by the same circuitry.

Here's another example showing +9 subtracted from −4:

```
  11100    (−4)
−01001     (+9)
```

Negate the subtrahend (+9) to produce 10111 (−9) and add this to the minuend (−4).

```
  11100    (−4)
+ 10111    (−9)
  10011    (−13)
↑
└ Disregard
```

The reader should verify the results of using the above procedure for the following subtractions: (a) +9 − (−4); (b) −9 − (+4); (c) −9 − (−4); (d) +4 − (−4). Remember that when the result has a sign bit of 1, it is negative and in 2's-complement form.

Arithmetic Overflow

In each of the previous addition and subtraction examples, the numbers that were added consisted of a sign bit and four magnitude bits. The answers also consisted of a sign bit and four magnitude bits. Any carry into the sixth bit position was disregarded. In all of the cases considered, the magnitude of the answer was small enough to fit into four bits. Let's look at the addition of +9 and +8.

```
+9 →   0   1001
+8 →   0   1000
       1   0001
       ↑       ↑
incorrect sign   incorrect magnitude
```

The answer has a negative sign bit, which is obviously incorrect since we are adding two positive numbers. The answer should be +17, but the magnitude 17 requires