ISBN/ISSN: 0132067110

straightforward implementation without directly solving this larger problem. This is an example of the power of iterative circuits and circuit reuse in design.

4-3 Binary Subtraction

In Chapter 1, we briefly examined the subtraction of unsigned binary numbers. Although beginning texts cover only signed-number addition and subtraction, to the complete exclusion of the unsigned alternative, unsigned-number arithmetic plays an important role in computation and computer hardware design. It is used in floating-point units, in signed-magnitude addition and subtraction algorithms, and in extending the precision of fixed-point numbers. For these reasons, we will treat unsigned-number addition and subtraction here. We also, however, choose to treat it first so that we can clearly justify, in terms of hardware cost, that which otherwise appears bizarre and often is accepted on faith, namely, the use of complement representations in arithmetic.

In Section 1-3, subtraction is performed by comparing the subtrahend with the minuend and subtracting the smaller from the larger. The use of a method containing this comparison operation results in inefficient and costly circuitry. As an alternative, we can simply subtract the subtrahend from the minuend. Using the same numbers as in a subtraction example from Section 1-3, we have

Borrows into:	11100
Minuend:	10011
Subtrahend:	−11110
Difference:	10101
Correct Difference:	−01011

If no borrow occurs into the most significant position, then we know that the subtrahend is not larger than the minuend and that the result is positive and correct. If a borrow does occur into the most significant position, as indicated in blue, then we know that the subtrahend is larger than the minuend. The result must then be negative, and so we need to correct its magnitude. We can do this by examining the result of the calculation when a borrow occurs:

$$M - N + 2^n$$

Note that the added 2^n represents the value of the borrow into the most significant position. Instead of this result, the desired magnitude is $N - M$. This can be obtained by subtracting the preceding formula from 2^n:

$$2^n - (M - N + 2^n) = N - M$$

In the previous example, 100000 − 10101 = 01011, which is the correct magnitude.

In general, the subtraction of two n-digit numbers, $M - N$, in base 2 can be done as follows:

1. Subtract the subtrahend N from the minuend M.
2. If no end borrow occurs, then $M \geq N$, and the result is nonnegative and correct.
3. If an end borrow occurs, then $N > M$, and the difference, $M - N + 2^n$, is subtracted from 2^n, and a minus sign is appended to the result.

Subtraction of a binary number from 2^n to obtain an n-digit result is called taking the *2s complement* of the number. So in step 3, we are taking the 2s complement of the difference $M - N + 2^n$. Use of the 2s complement in subtraction is illustrated by the following example.

EXAMPLE 4-1 Unsigned Binary Subtraction by 2s Complement Subtract

Perform the binary subtraction 01100100 − 10010110. We have

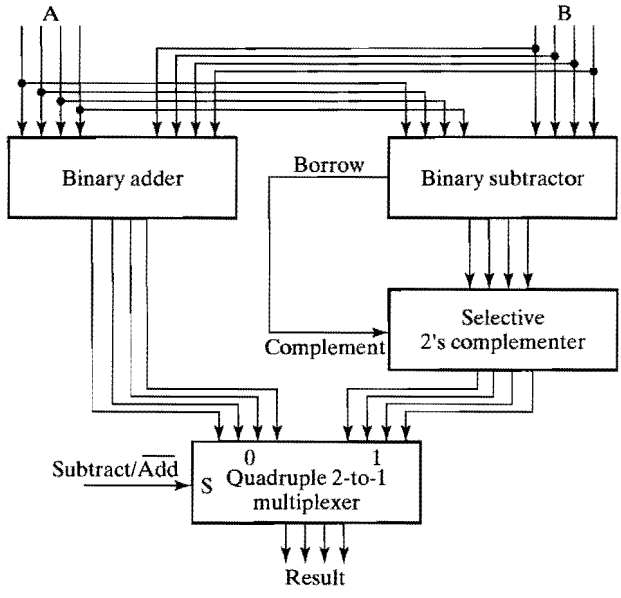
Borrows into:	10011110
Minuend:	01100100
Subtrahend:	− 10010110
Initial Result	11001110

The end borrow of 1 implies correction:

2^8	100000000
− Initial Result	− 11001110
Final Result	− 00110010

■

To perform subtraction using this method requires a subtractor for the initial subtraction. In addition, when necessary, either the subtractor must be used a second time to perform the correction, or a separate 2s complementer circuit must be provided. So, thus far, we require a subtractor, an adder, and possibly a 2s complementer to perform both addition and subtraction. The block diagram for a 4-bit adder–subtractor using these functional blocks is shown in Figure 4-6. The inputs are applied to both the adder and the subtractor, so both operations are performed in parallel. If an end borrow value of 1 occurs in the subtraction, then the selective 2s complementer receives a value of 1 on its *Complement* input. This circuit then takes the 2s complement of the output of the subtractor. If the end borrow has value of 0, the selective 2s complementer passes the output of the subtractor through unchanged. If subtraction is the operation, then a 1 is applied to S of the multiplexer that selects the output of the complementer. If addition is the operation, then a 0 is applied to S, thereby selecting the output of the adder.



□ **FIGURE 4-6**
Block Diagram of Binary Adder–Subtractor

As we will see, this circuit is more complex than necessary. To reduce the amount of hardware, we would like to share logic between the adder and the subtractor. This can also be done using the notion of the complement. So before considering the combined adder–subtractor further, we will take a more careful look at complements.

Complements

There are two types of complements for each base-r system: the *radix complement*, which we saw earlier for base 2, and the *diminished radix complement*. The first is referred to as the *r's complement* and the second as the *(r − 1)'s complement*. When the value of the base r is substituted in the names, the two types are referred to as the 2s and 1s complements for binary numbers and the 10s and 9s complements for decimal numbers, respectively. Since our interest for the present is in binary numbers and operations, we will deal with only 1s and 2s complements.

Given a number N in binary having n digits, the *1s complement of N* is defined as $(2^n - 1) - N$. 2^n is represented by a binary number that consists of a 1 followed by n 0s. $2^n - 1$ is a binary number represented by n 1s. For example, if $n = 4$, we have $2^4 = (10000)_2$ and $2^4 - 1 = (1111)_2$. Thus, the 1s complement of a binary number is obtained by subtracting each digit from 1. When subtracting binary digits from 1, we can have either $1 - 0 = 1$ or $1 - 1 = 0$, which

causes the original bit to change from 0 to 1 or from 1 to 0, respectively. Therefore, the 1s complement of a binary number is formed by changing all 1s to 0s and all 0s to 1s—that is, applying the NOT or complement operation to each of the bits. Following are two numerical examples:

The 1's complement of 1011001 is 0100110.
The 1's complement of 0001111 is 1110000.

In similar fashion, the 9s complement of a decimal number, the 7's complement of an octal number, and the 15s complement of a hexadecimal number are obtained by subtracting each digit from 9, 7, and F (decimal 15), respectively.

Given an n-digit number N in binary, the *2s complement of* N is defined as $2^n - N$ for $N \neq 0$ and 0 for $N = 0$. The reason for the special case of $N = 0$ is that the result must have n bits, and subtraction of 0 from 2^n gives an $(n + 1)$-bit result, 100...0. This special case is achieved by using only an n-bit subtractor or otherwise dropping the 1 in the extra position. Comparing with the 1s complement, we note that the 2s complement can be obtained by adding 1 to the 1s complement, since $2^n - N = \{[(2^n - 1) - N] + 1\}$. For example, the 2s complement of binary 101100 is $010011 + 1 = 010100$ and is obtained by adding 1 to the 1s complement value. Again, for $N = 0$, the result of this addition is 0, achieved by ignoring the carry out of the most significant position of the addition. These concepts hold for other bases as well. As we will see later, they are very useful in simplifying 2s complement and subtraction hardware.

Also, the 2s complement can be formed by leaving all least significant 0s and the first 1 unchanged and then replacing 1s with 0s and 0s with 1s in all other higher significant bits. Thus, the 2s complement of 1101100 is 0010100 and is obtained by leaving the two low-order 0s and the first 1 unchanged and then replacing 1s with 0s and 0s with 1s in the other four most significant bits. In other bases, the first nonzero digit is subtracted from the base r, and the remaining digits to the left are replaced with $r - 1$ minus their values.

It is also worth mentioning that the complement of the complement restores the number to its original value. To see this, note that the 2s complement of N is $2^n - N$, and the complement of the complement is $2^n - (2^n - N) = N$, giving back the original number.

Subtraction Using 2s Complement

Earlier, we expressed a desire to simplify hardware by sharing adder and subtractor logic. Armed with complements, we are prepared to define a binary subtraction procedure that uses addition and the corresponding complement logic. The subtraction of two n-digit unsigned numbers, $M - N$, in binary can be done as follows:

1. Add the 2s complement of the subtrahend N to the minuend M. This performs $M + (2^n - N) = M - N + 2^n$.

2. If $M \geq N$, the sum produces an end carry, 2^n. Discard the end carry, leaving result $M - N$.
3. If $M < N$, the sum does not produce an end carry, since it is equal to $2^n - (N - M)$, the 2s complement of $N - M$. Perform a correction, taking the 2s complement of the sum and placing a minus sign in front to obtain the result $-(N - M)$.

The examples that follow further illustrate the foregoing procedure. Note that, although we are dealing with unsigned numbers, there is no way to get an unsigned result for the case in step 3. When working with paper and pencil, we recognize, by the absence of the end carry, that the answer must be changed to a negative number. If the minus sign for the result is to be preserved, it must be stored separately from the corrected n-bit result.

EXAMPLE 4-2 Unsigned Binary Subtraction by 2s Complement Addition

Given the two binary numbers $X = 1010100$ and $Y = 1000011$, perform the subtraction $X - Y$ and $Y - X$ using 2s complement operations. We have

$$\begin{aligned} X &= \quad 1010100 \\ \text{2s complement of } Y &= \quad 0111101 \\ \text{Sum} &= \quad 10010001 \\ \text{Discard end carry } 2^7 &= -\underline{10000000} \\ \textit{Answer: } X - Y &= \quad 0010001 \end{aligned}$$

$$\begin{aligned} Y &= \quad 1000011 \\ \text{2s complement of } X &= \quad \underline{0101100} \\ \text{Sum} &= \quad 1101111 \end{aligned}$$

There is no end carry.

Answer: $Y - X = -(\text{2s complement of } 1101111) = -0010001$. ■

While subtraction of unsigned numbers also can be done by means of the 1s complement, it is little used in modern designs, so will not be covered here.

4-4 Binary Adder–Subtractors

Using the 2s complement, we have eliminated the subtraction operation and need only the complementer and an adder. When performing a subtraction we complement the subtrahend N, and when performing an addition we do not complement N. These operations can be accomplished by using a selective complementer and