The notation 4{B[3]} denotes that the bit B[3] is replicated four times; it is equivalent to writing {B[3], B[3], B[3], B[3]}. This is referred to as the *replication* operator, which is discussed in section 6.6.5 in Chapter 6. If we want to generate a carry-out signal from bit position 7, then we could adopt the approach in Figure 5.32 by using the statement

$$C = \{1'b0, A\} + \{1'b0, 4\{B[3]\}, B\};$$

5.6 MULTIPLICATION

Before we discuss the general issue of multiplication, we should note that a binary number, B, can be multiplied by 2 simply by adding a zero to the right of its least-significant bit. This effectively moves all bits of B to the left, and we say that B is *shifted* left by one bit position. Thus if $B = b_{n-1}b_{n-2}\cdots b_1b_0$, then $2 \times B = b_{n-1}b_{n-2}\cdots b_1b_0 0$. (We have already used this fact in section 5.2.3.) Similarly, a number is multiplied by 2^k by shifting it left by k bit positions. This is true for both unsigned and signed numbers.

We should also consider what happens if a binary number is shifted right by k bit positions. According to the positional number representation, this action divides the number by 2^k. For unsigned numbers the shifting amounts to adding k zeros to the left of the most-significant bit. For example, if B is an unsigned number, then $B \div 2 = 0b_{n-1}b_{n-2}\cdots b_2b_1$. Note that bit b_0 is lost when shifting to the right. For signed numbers it is necessary to preserve the sign. This is done by shifting the bits to the right and filling from the left with the value of the sign bit. Hence if B is a signed number, then $B \div 2 = b_{n-1}b_{n-1}b_{n-2}\cdots b_2b_1$. For instance, if $B = 011000 = (24)_{10}$, then $B \div 2 = 001100 = (12)_{10}$ and $B \div 4 = 000110 = (6)_{10}$. Similarly, if $B = 101000 = -(24)_{10}$, then $B \div 2 = 110100 = -(12)_{10}$ and $B \div 4 = 111010 = -(6)_{10}$. The reader should also observe that the smaller the positive number, the more 0s there are to the left of the first 1, while for a negative number there are more 1s to the left of the first 0.

Now we can turn our attention to the general task of multiplication. Two binary numbers can be multiplied using the same method as we use for decimal numbers. We will focus our discussion on multiplication of unsigned numbers. Figure 5.35*a* shows how multiplication is performed manually, using four-bit numbers. Each multiplier bit is examined from right to left. If a bit is equal to 1, an appropriately shifted version of the multiplicand is added to form a *partial product*. If the multiplier bit is equal to 0, then nothing is added. The sum of all shifted versions of the multiplicand is the desired product. Note that the product occupies eight bits.

The same scheme can be used to design a multiplier circuit. We will stay with four-bit numbers to keep the discussion simple. Let the multiplicand, multiplier, and product be denoted as $M = m_3m_2m_1m_0$, $Q = q_3q_2q_1q_0$, and $P = p_7p_6p_5p_4p_3p_2p_1p_0$, respectively. One simple way of implementing the multiplication scheme is to use a sequential approach, where an eight-bit adder is used to compute partial products. As a first step, the bit q_0 is examined. If $q_0 = 1$, then M is added to the initial partial product, which is initialized to 0. If $q_0 = 0$, then 0 is added to the partial product. Next q_1 is examined. If $q_1 = 1$, then the value $2 \times M$ is added to the partial product. The value $2 \times M$ is created simply by

Multiplicand M	(14)	1 1 1 0
Multiplier Q	(11)	× 1 0 1 1
		1 1 1 0
		1 1 1 0
		0 0 0 0
		1 1 1 0
Product P	(154)	1 0 0 1 1 0 1 0

(a) Multiplication by hand

Multiplicand M	(11)	1 1 1 0
Multiplier Q	(14)	× 1 0 1 1
Partial product 0		1 1 1 0
		+ 1 1 1 0
Partial product 1		1 0 1 0 1
		+ 0 0 0 0
Partial product 2		0 1 0 1 0
		+ 1 1 1 0
Product P	(154)	1 0 0 1 1 0 1 0

(b) Multiplication for implementation in hardware

Figure 5.35 Multiplication of unsigned numbers.

shifting M one bit position to the left. Similarly, $4 \times M$ is added to the partial product if $q_2 = 1$, and $8 \times M$ is added if $q_3 = 1$. We will show in Chapter 10 how such a circuit may be implemented.

This sequential approach leads to a relatively slow circuit, primarily because a single eight-bit adder is used to perform all additions needed to generate the partial products and the final product. A much faster circuit can be obtained if multiple adders are used to compute the partial products.

5.6.1 ARRAY MULTIPLIER FOR UNSIGNED NUMBERS

Figure 5.35*b* indicates how multiplication may be performed by using multiple adders. In each step a four-bit adder is used to compute the new partial product. Note that as the computation progresses, the least-significant bits are not affected by subsequent additions; hence they can be passed directly to the final product, as indicated by blue arrows. Of course, these bits are a part of the partial products as well.

A fast multiplier circuit can be designed using an array structure that is similar to the organization in Figure 5.35*b*. Consider a 4 × 4 example, where the multiplicand and multiplier are $M = m_3m_2m_1m_0$ and $Q = q_3q_2q_1q_0$, respectively. The partial product 0, $PP0 = pp0_3\ pp0_2\ pp0_1\ pp0_0$, can be generated using the AND of q_0 with each bit of M. Thus

$$PP0 = m_3q_0\ \ m_2q_0\ \ m_1q_0\ \ m_0q_0$$

Partial product 1, $PP1$, is generated using the AND of q_1 with M and adding it to $PP0$ as follows

$PP0$:		0	$pp0_3$	$pp0_2$	$pp0_1$	$pp0_0$
	+	m_3q_1	m_2q_1	m_1q_1	m_0q_1	0
$PP1$:		$pp1_4$	$pp1_3$	$pp1_2$	$pp1_1$	$pp1_0$

Similarly, partial product 2, $PP2$, is generated using the AND of q_2 with M and adding to $PP1$, and so on.

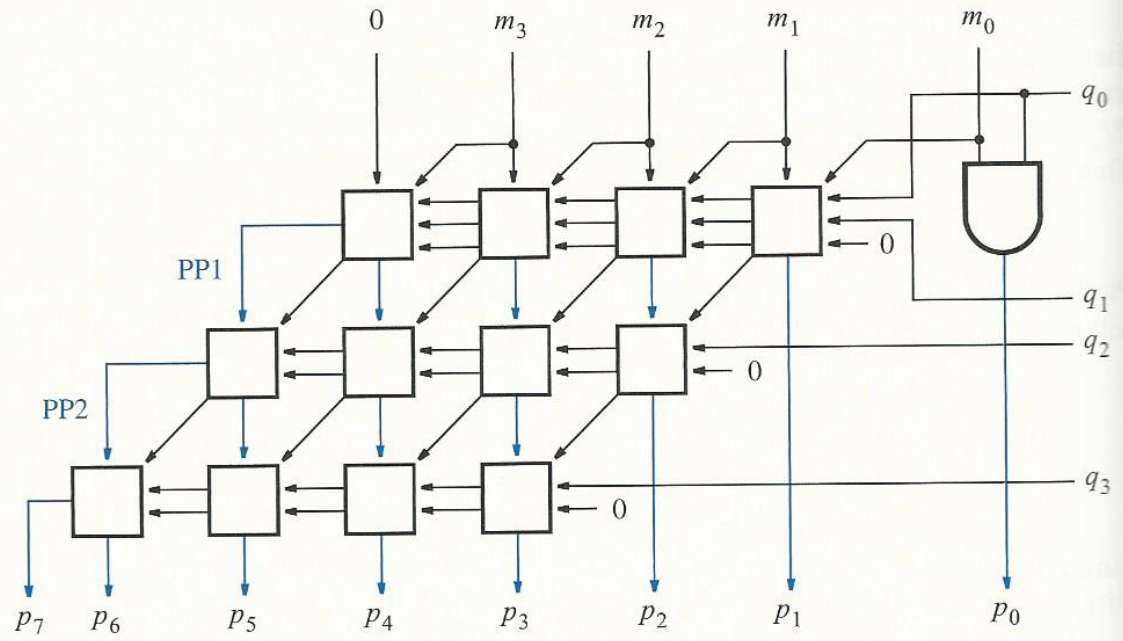
A circuit that implements the preceding operations is arranged in an array, as shown in Figure 5.36*a*. There are two types of blocks in the array. Part (*b*) of the figure shows the details of the blocks in the top row, and part (*c*) shows the block used in the second and third rows. Observe that the shifted versions of the multiplicand are provided by routing the m_k signals diagonally from one block to another. The full-adder included in each block implements a ripple-carry adder to generate each partial product. It is possible to design even faster multipliers by using other types of adders [1].

5.6.2 MULTIPLICATION OF SIGNED NUMBERS

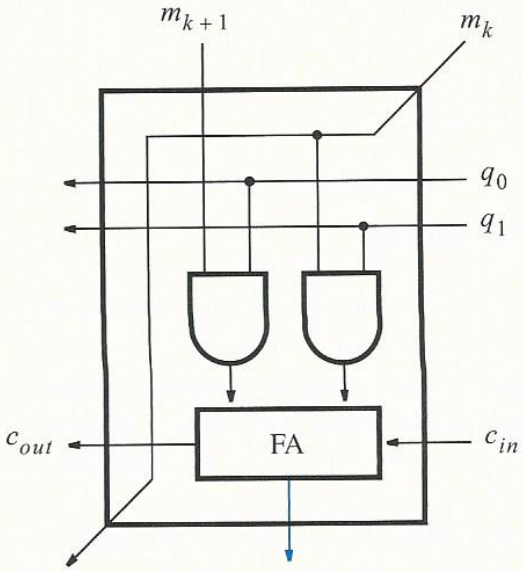
Multiplication of unsigned numbers illustrates the main issues involved in the design of multiplier circuits. Multiplication of signed numbers is somewhat more complex.

If the multiplier operand is positive, it is possible to use essentially the same scheme as for unsigned numbers. For each bit of the multiplier operand that is equal to 1, a properly shifted version of the multiplicand must be added to the partial product. The multiplicand can be either positive or negative.

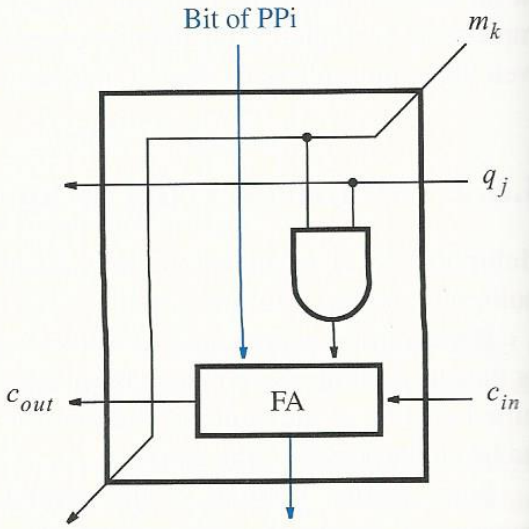
Since shifted versions of the multiplicand are added to the partial products, it is important to ensure that the numbers involved are represented correctly. For example, if the two right-most bits of the multiplier are both equal to 1, then the first addition must produce the partial product $PP1 = M + 2M$, where M is the multiplicand. If $M = m_{n-1}m_{n-2}\cdots m_1m_0$, then $PP1 = m_{n-1}m_{n-2}\cdots m_1m_0 + m_{n-1}m_{n-2}\cdots m_1m_00$. The adder that performs this addition comprises circuitry that adds two operands of equal length. Since shifting the multiplicand to the left, to generate $2M$, results in one of the operands having $n + 1$ bits, the required addition has to be performed using the second operand, M, represented also as an $(n + 1)$-bit number. An n-bit signed number is represented as an $(n + 1)$-bit number by using sign extension, that is, by replicating the sign bit as the new left-most bit. Thus $M = m_{n-1}m_{n-2}\cdots m_1m_0$ is represented using $(n + 1)$ bits as $M = m_{n-1}m_{n-1}m_{n-2}\cdots m_1m_0$.



(a) Structure of the circuit



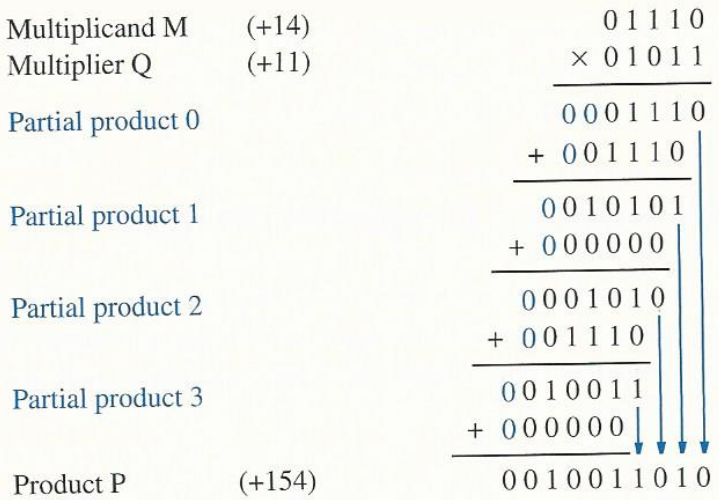
(b) A block in the top row



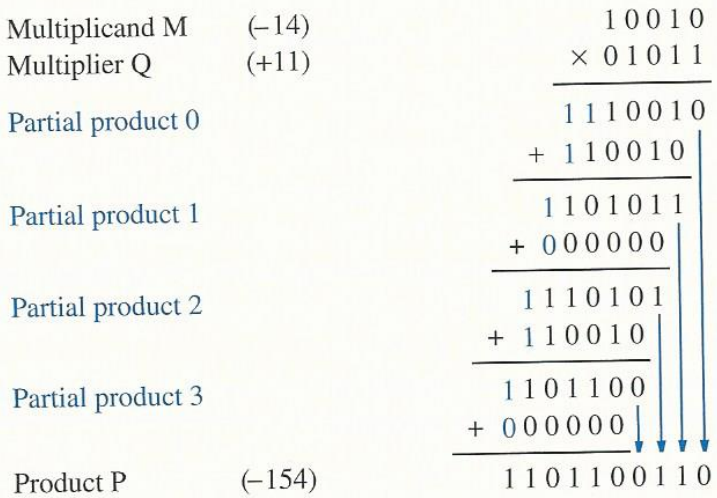
(c) A block in the bottom two rows

Figure 5.36 A 4 × 4 multiplier circuit.

When a shifted version of the multiplicand is added to a partial product, overflow has to be avoided. Hence the new partial product must be larger by one extra bit. Figure 5.37*a* illustrates the process of multiplying two positive numbers. The sign-extended bits are shown in blue. Part (*b*) of the figure involves a negative multiplicand. Note that the resulting product has $2n$ bits in both cases.



(a) Positive multiplicand



(b) Negative multiplicand

Figure 5.37 Multiplication of signed numbers.

For a negative multiplier operand, it is possible to convert both the multiplier and the multiplicand into their 2's complements because this will not change the value of the result. Then the scheme for a positive multiplier can be used.

We have presented a relatively simple scheme for multiplication of signed numbers. There exist other techniques that are more efficient but also more complex. We will not pursue these techniques, but an interested reader may consult reference [1].

We have discussed circuits that perform addition, subtraction, and multiplication. Another arithmetic operation that is needed in computer systems is division. Circuits that perform division are more complex; we will present an example in Chapter 10. Various

techniques for performing division are usually discussed in books on the subject of computer organization, and can be found in references [1, 2].