Chapter 4

Operations on Bits

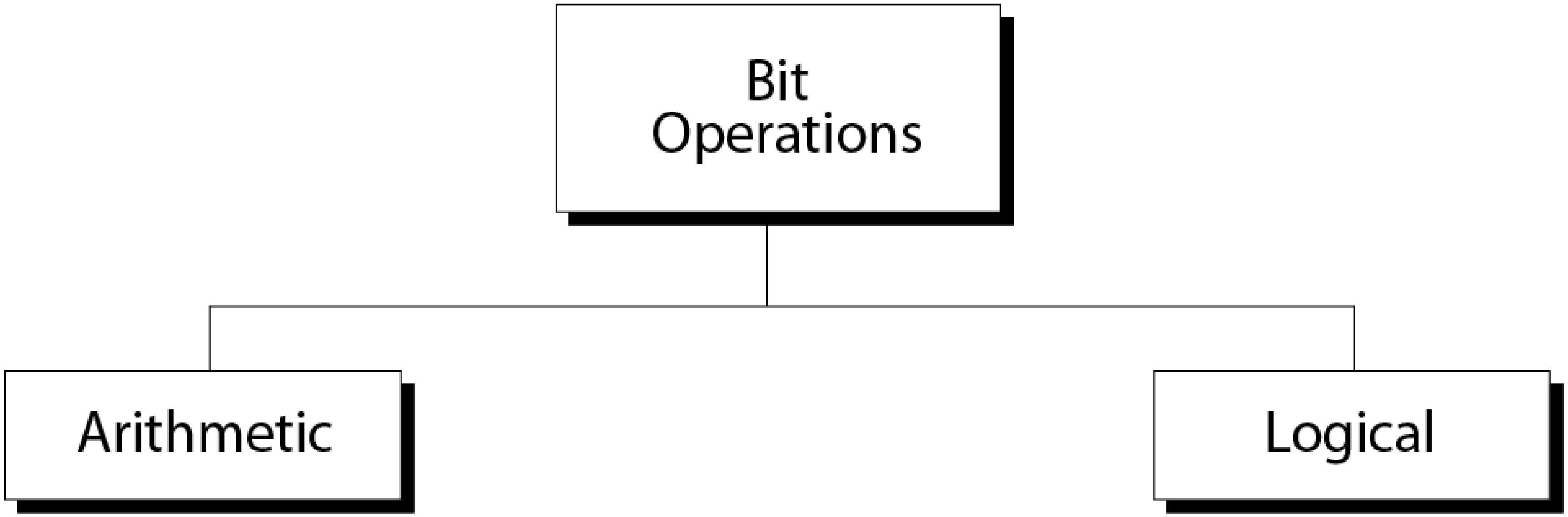
OBJECTIVES

After reading this chapter, the reader should be able to:

- Apply arithmetic operations on bits when the integer is represented in two’s complement.
- Apply logical operations on bits.
- Understand the applications of logical operations using masks.
- Understand the shift operations on numbers and how a number can be multiplied or divided by powers of two using shift operations.

Figure 4-1

Operations on bits



4.1

ARITHMETIC OPERATIONS

Table 4.1 Adding bits

Number of 1s	*Result*	*Carry*
[redacted]	[redacted]	------------ 1 1

Note:

Rule of Adding Integers in Two's Complement

Add 2 bits and propagate the carry to the next column. If there is a final carry after the leftmost column addition, discard it.

Example 1

Add two numbers in two's complement representation: (+17) + (+22) ➔ (+39)

Solution

Carry			1							
	0	0	0	1	0	0	0	1	+	
	0	0	0	1	0	1	1	0		

Result	0	0	1	0	0	1	1	1	➔	39

Example 2

Add two numbers in two's complement representation: (+24) + (-17) ➔ (+7)

Solution

Carry	1	1	1	1	1					
	0	0	0	1	1	0	0	0	+	
	1	1	1	0	1	1	1	1		

Result	0	0	0	0	0	1	1	1	➔	+7

Example 3

Add two numbers in two's complement representation: (-35) + (+20) ➔ (-15)

Solution

Carry			1	1	1					
	1	1	0	1	1	1	0	1	+	
	0	0	0	1	0	1	0	0		

Result	1	1	1	1	0	0	0	1	➔	-15

簡要解釋爲何two’s complement 可以這樣做運算

- 假設是*n* bits
- 正數 + 正數 (和一般情況一樣)
- 負數(-x) + 負數(-y)
  -x在two’s complement表示值爲 2n-x
  -y在two’s complement表示值爲 2n-y

  2n - x + 2n - y = 2n + (2n - (x+y))

Carry (進位)

-(x+y)的two’s complement表示法

簡要解釋爲何two’s complement 可以這樣做運算 (續前頁)

- 正數 (x) + 負數 (-y)
  -y在two’s complement表示值爲 2n-y
  得 2n+x-y
  (1) x >= y
  x-y爲正值; 2n爲進位
- (2) x <y
- 2n+x-y = 2n-(y-x)

Example 4

Add two numbers in two's complement representation: (+127) + (+3) ➔ (+130)

Solution

Carry	1	1	1	1	1	1	1		
	0	1	1	1	1	1	1	1	+
	0	0	0	0	0	0	1	1	

Result	1	0	0	0	0	0	1	0	➔ ***-126 (Error)***

An overflow has occurred.

Note:

Range of numbers in two's complement representation

$-(2^{N-1})$ ---------- 0 ----------- $+(2^{N-1}-1)$

Note:

When you do arithmetic operations on numbers in a computer, remember that each number and the result should be in the range defined by the bit allocation.

Overflow Detection

- Consider -7+ -2

Overflow when these two bits are not equal

- 1 0 0 0 (Carry bits)
- 1 0 0 1 (-7)
- + 1 1 1 0 (-2)

- 1 0 1 1 1

Ignore last carry bit

- -7+(-2)=-9

Example 5

Subtract 62 from 101 in two's complement:

(+101) - (+62) ⇔ (+101) + (-62)

Solution

Carry	1	1								
	0	1	1	0	0	1	0	1	+	
	1	1	0	0	0	0	1	0		

Result	0	0	1	0	0	1	1	1	→	39

The leftmost carry is discarded.

Example 6

Add two floats:

0 10000100 10110000000000000000000

0 10000010 01100000000000000000000

Solution

The exponents are 5 and 3. The numbers are:

$+2^5$ x 1.1011   and $+2^3$ x 1.011

Make the exponents the same.

($+2^5$ x 1.1011)+ ($+2^5$ x 0.01011) ➔ $+2^5$ x 10.00001

After normalization $+2^6$ x 1.000001, which is stored as:

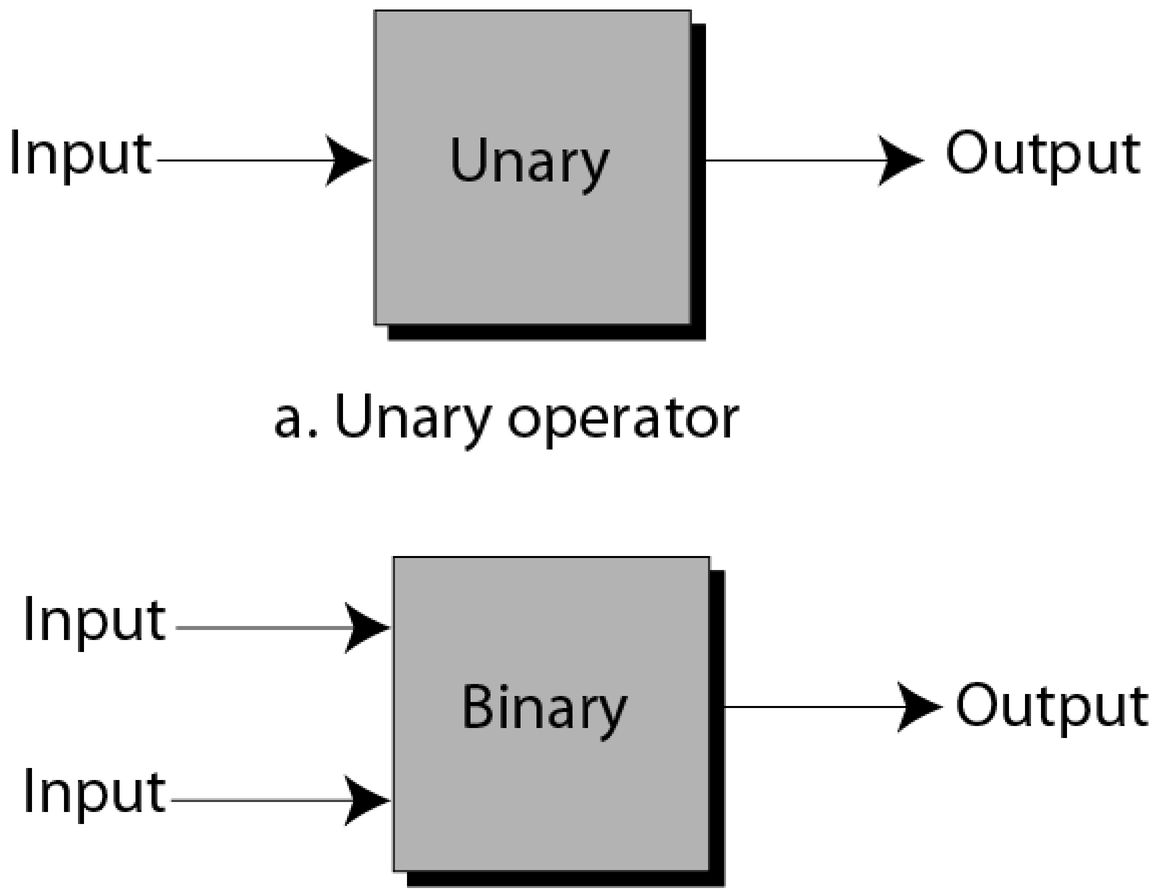
0 10000101 00000100000000000000000

4.2

LOGICAL OPERATIONS

Figure 4-3

Unary and binary operations

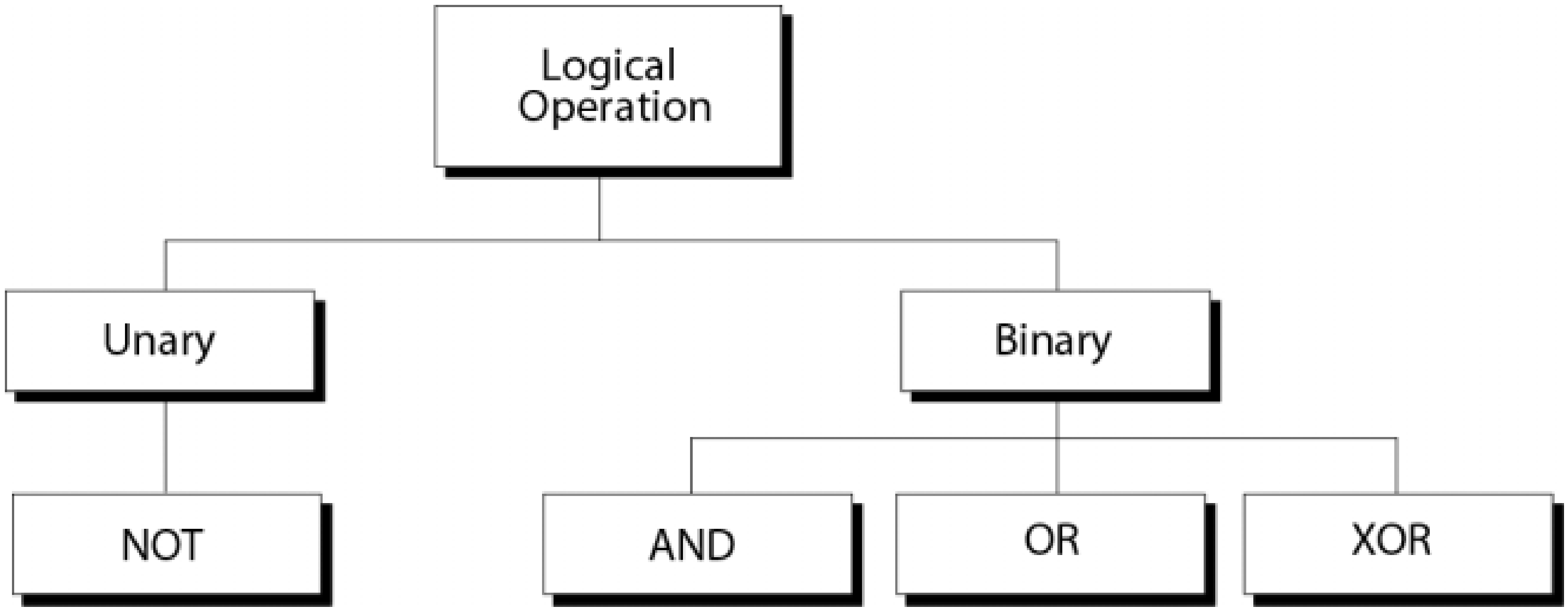


a. Unary operator

b. Binary operator

Figure 4-4

Logical operations



Conventions for Boolean Algebra

- Conventions:
  - 1=True, 0=False
  - H=> (High voltage), L=>(Low voltage)
  - Logic convention
    - Positive logic convention
      - H=1=T, L=0=F
    - Negative logic convention
      - H=0=F, L=1=T

Figure 4-5

Truth tables

NOT

x	NOT x
0	1
1	0

AND

x	y	x AND y
0	0	0
0	1	0
1	0	0
1	1	1

OR

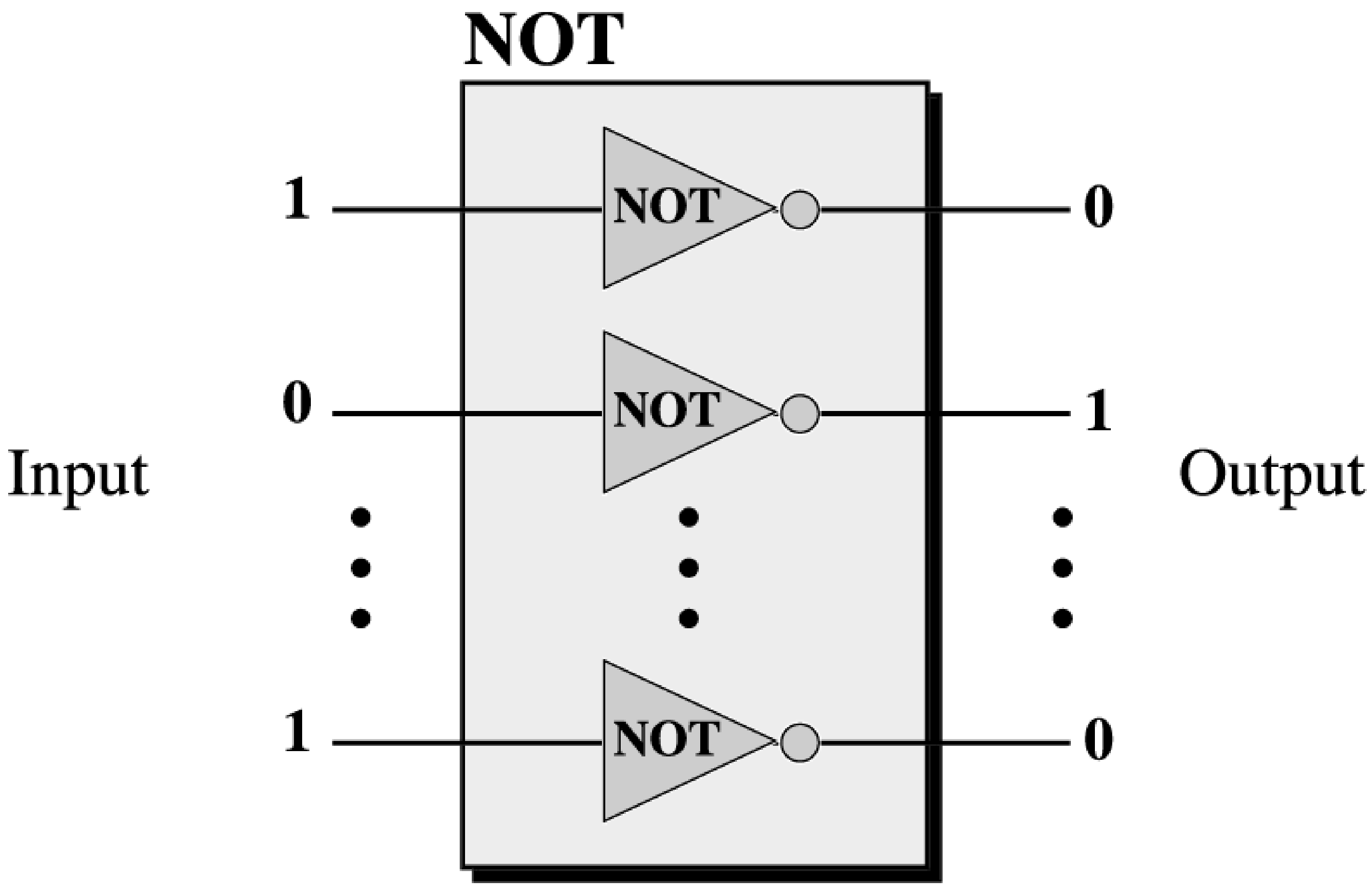
x	y	x OR y
0	0	0
0	1	1
1	0	1
1	1	1

XOR

x	y	x XOR y
0	0	0
0	1	1
1	0	1
1	1	0

Figure 4-6

NOT operator



Example 7

Use the NOT operator on the bit pattern 10011000

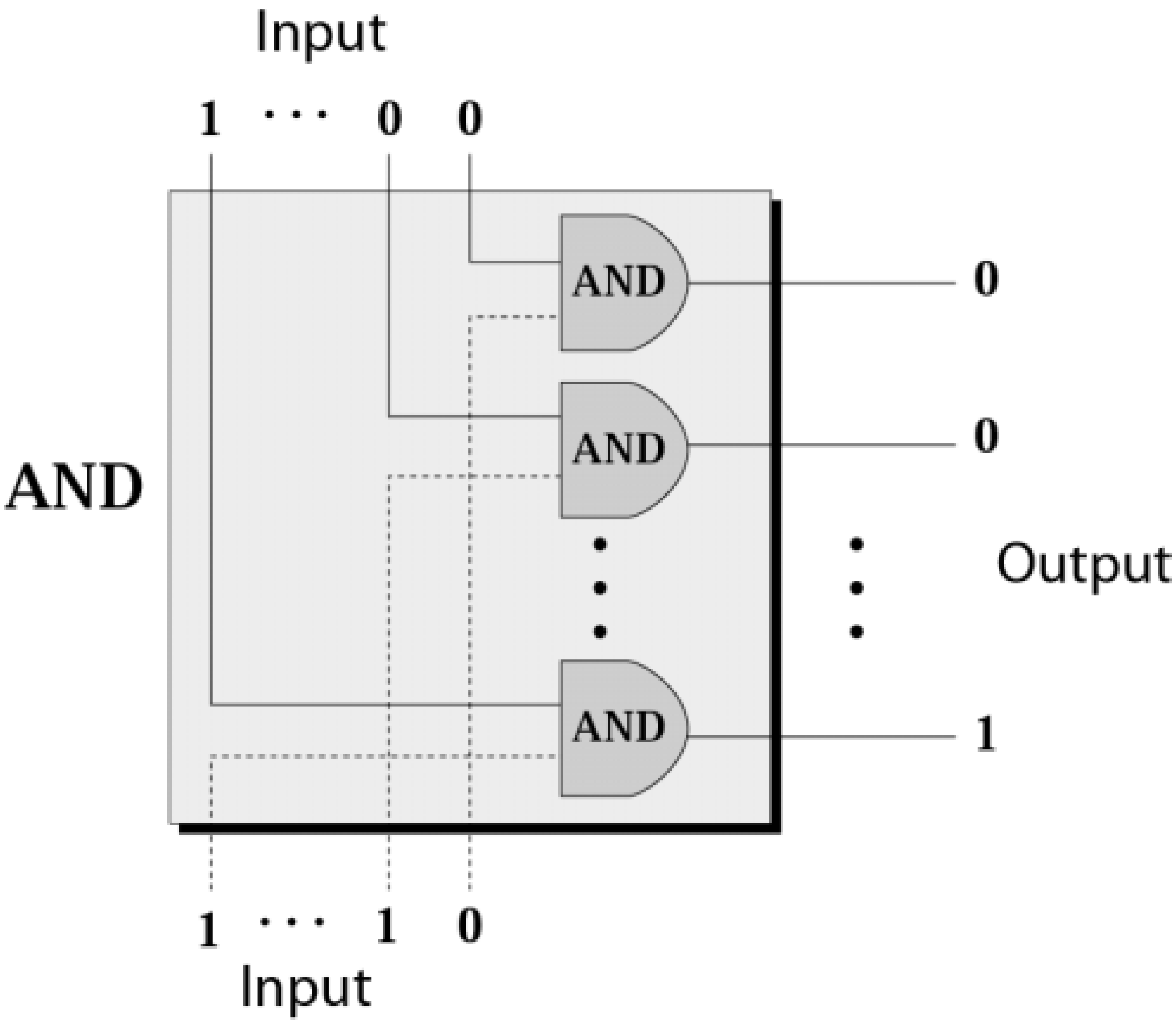
Solution

Target	**1 0 0 1 1 0 0 0**	***NOT***

Result	**0 1 1 0 0 1 1 1**	

Figure 4-7

AND operator



Example 8

Use the AND operator on bit patterns 10011000 and 00110101.

Solution

Target	**1 0 0 1 1 0 0 0**	***AND***
	0 0 1 1 0 1 0 1	

Result	**0 0 0 1 0 0 0 0**	

Figure 4-8

Inherent rule of the AND operator

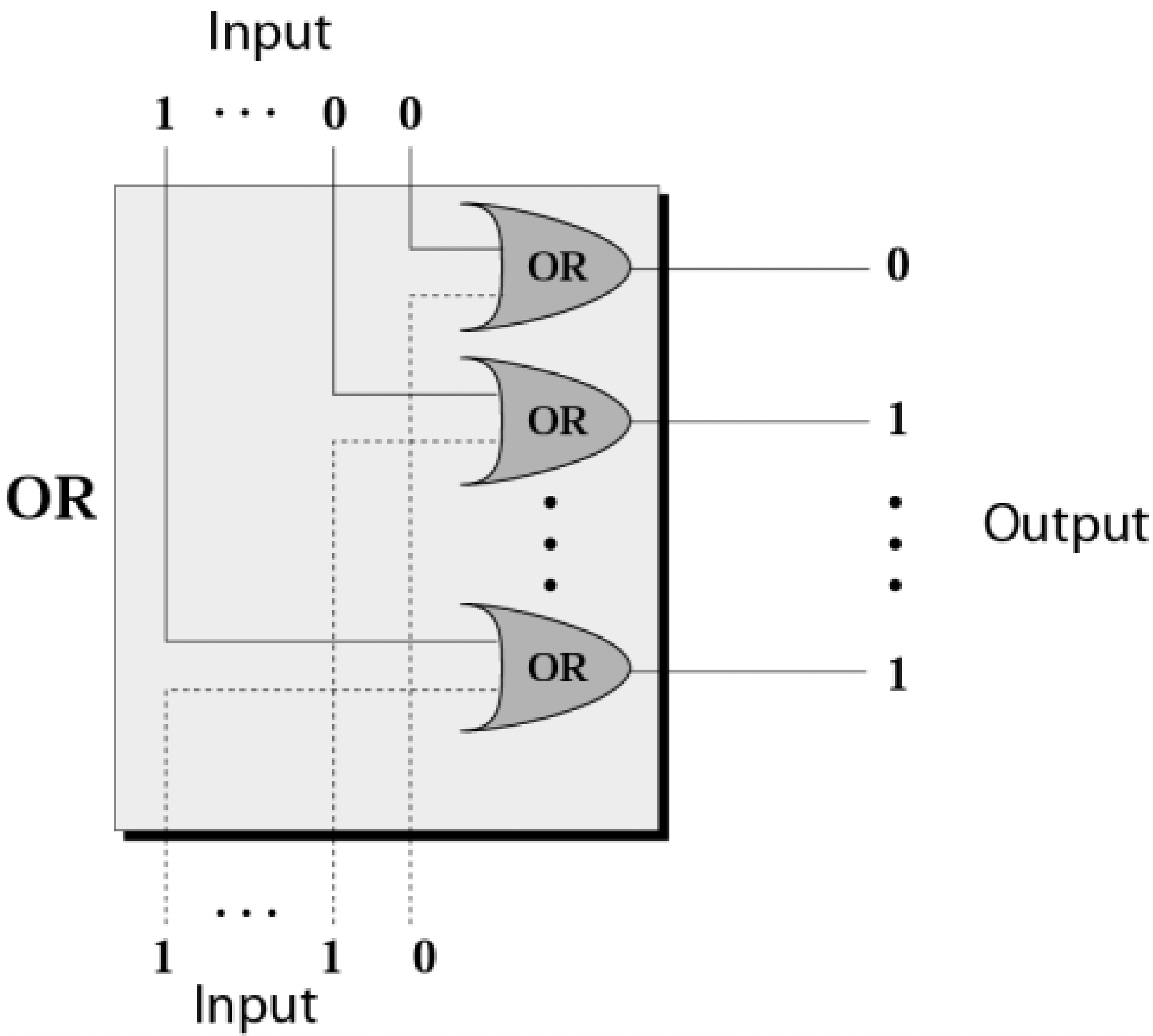
If a bit in one input is zero, then the result is zero.

$$(0) \quad \text{AND} \quad (X) \longrightarrow (0)$$

$$(X) \quad \text{AND} \quad (0) \longrightarrow (0)$$

Figure 4-9

OR operator



Example 9

Use the OR operator on bit patterns 10011000 and 00110101

Solution

Target	**1 0 0 1 1 0 0 0**	***OR***
	0 0 1 1 0 1 0 1	

Result	**1 0 1 1 1 1 0 1**	

Figure 4-10

Inherent rule of the OR operator

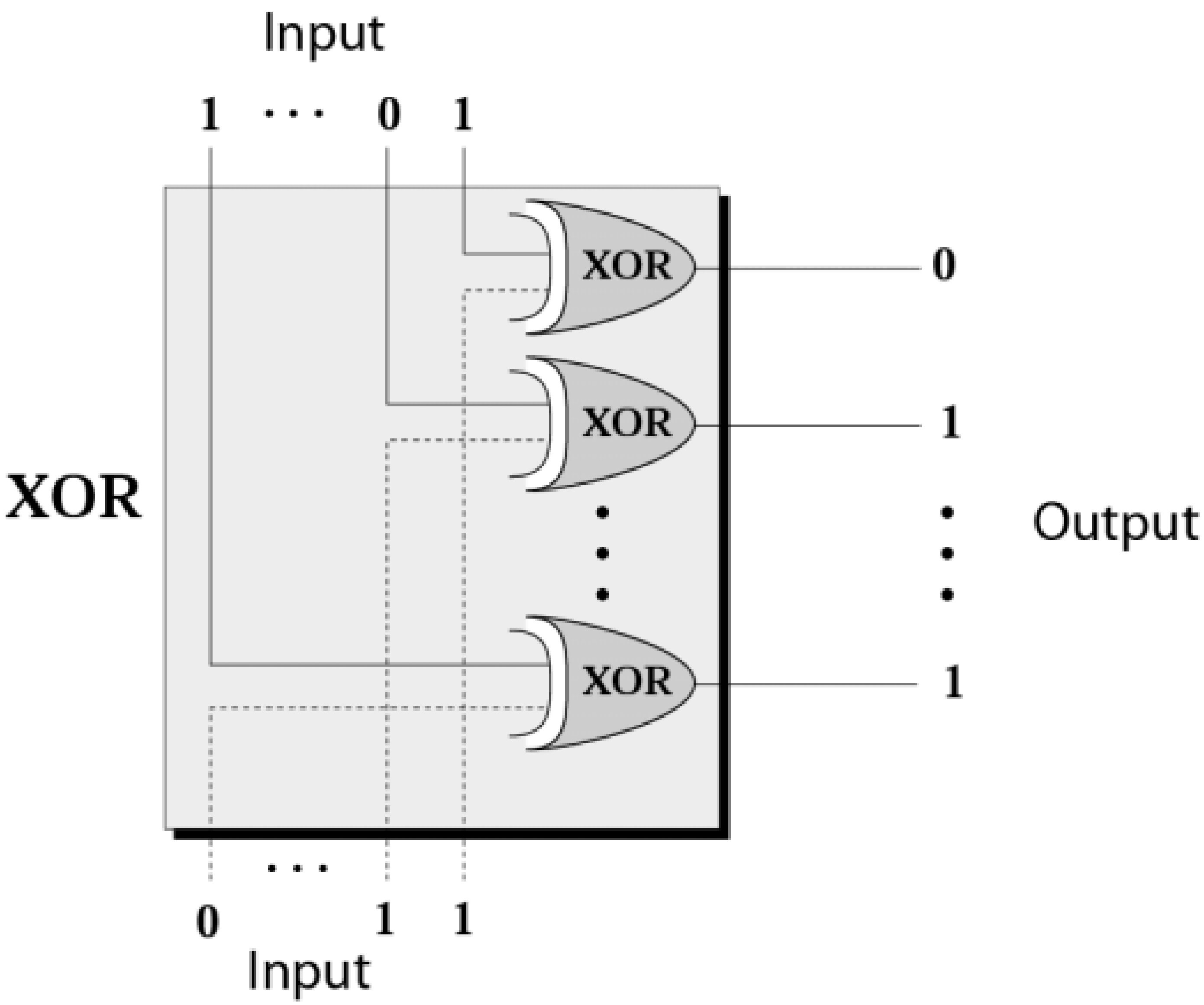
If a bit in one input is 1, then the result is 1.

(1) OR (X) $\longrightarrow$ (1)

(X) OR (1) $\longrightarrow$ (1)

Figure 4-11

XOR operator



Example 10

Use the XOR operator on bit patterns 10011000 and 00110101.

Solution

Target	**1 0 0 1 1 0 0 0**	***XOR***
	0 0 1 1 0 1 0 1	

Result	**1 0 1 0 1 1 0 1**	

Figure 4-12

Inherent rule of the XOR operator

$$(1) \quad \text{XOR} \quad (X) \longrightarrow \text{NOT}\,(X)$$

$$(X) \quad \text{XOR} \quad (1) \longrightarrow \text{NOT}\,(X)$$

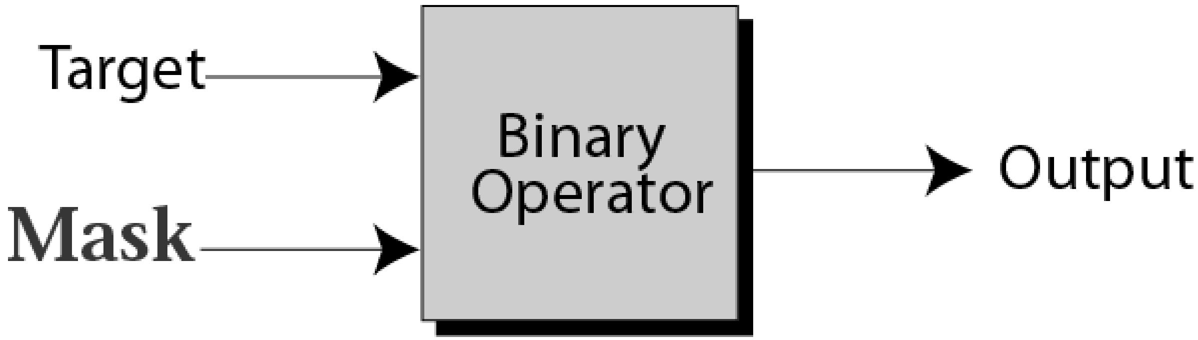
More about XOR

- 一連串的bits做 XOR, 若奇數個1, 則結果爲1; 若偶數個1則結果爲0

	1
	1
	0
	1
XOR	0
	1

Figure 4-13

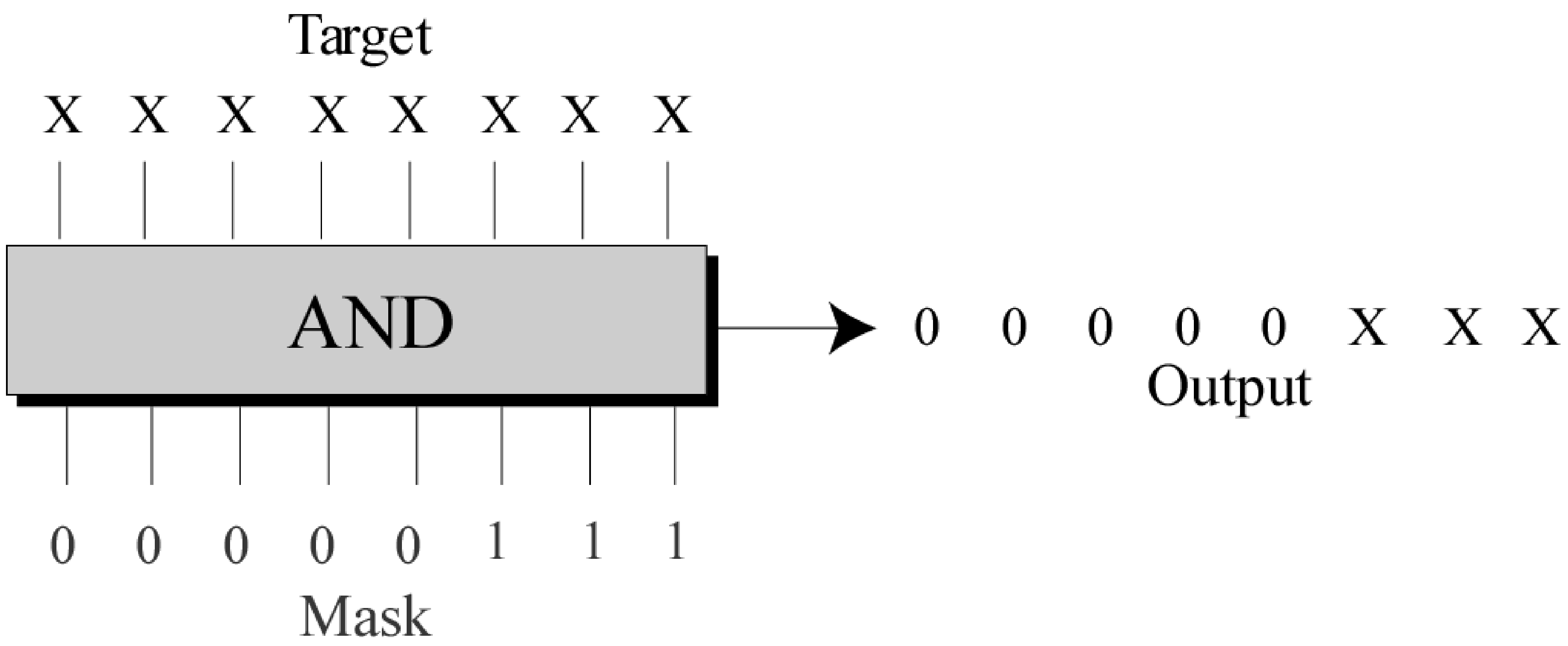
Mask



Use Mask to unset, set, or reverse the bit by ANDed, ORed, and XORed.

Figure 4-14

Example of unsetting specific bits



Example 11

Use a mask to unset (clear) the 5 leftmost bits of a pattern. Test the mask with the pattern 10100110.

Solution

The mask is 00000111.

Target	**1 0 1 0 0 1 1 0**	***AND***
Mask	**0 0 0 0 0 1 1 1**	

Result	**0 0 0 0 0 1 1 0**	

Example 12

Imagine a power plant that pumps water to a city using eight pumps. The state of the pumps (on or off) can be represented by an 8-bit pattern. For example, the pattern 11000111 shows that pumps 1 to 3 (from the right), 7 and 8 are on while pumps 4, 5, and 6 are off. Now assume pump 7 shuts down. How can a mask show this situation?

Solution on the next slide.

Solution

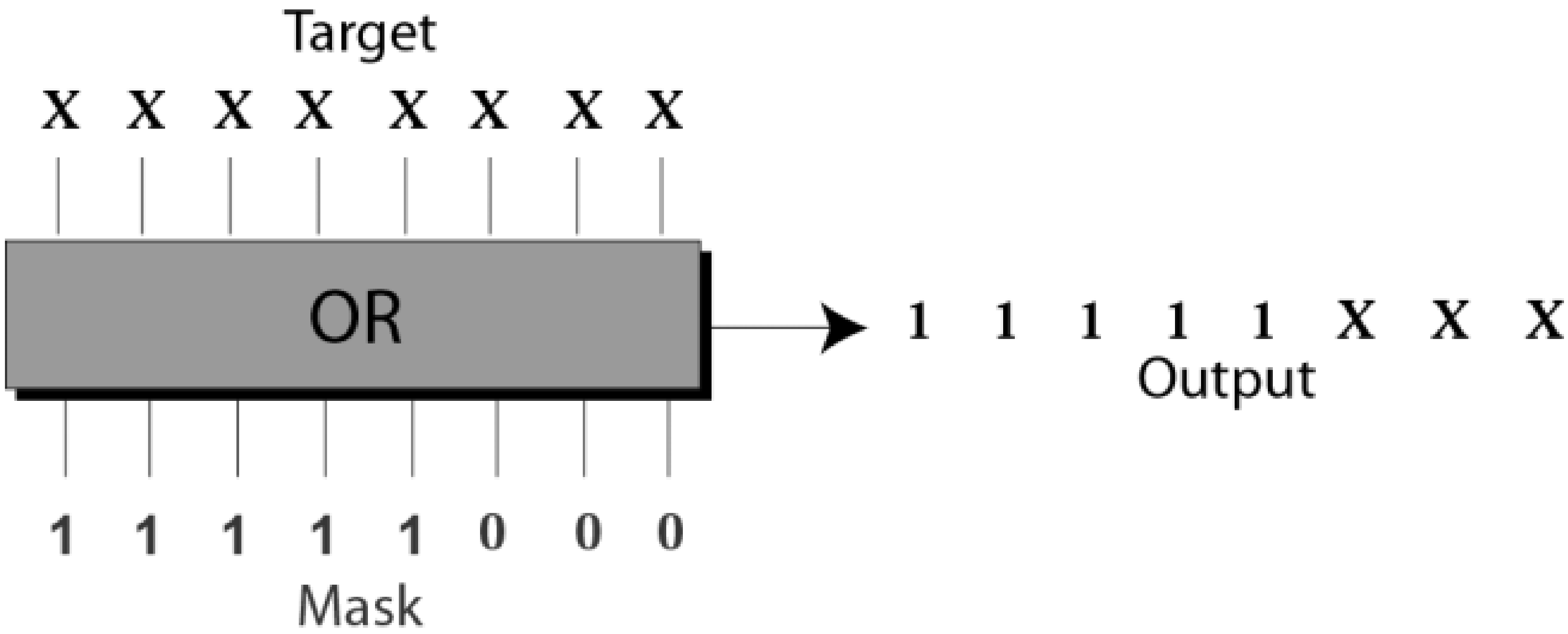
Use the mask 10111111 to AND with the target pattern. The only 0 bit (bit 7) in the mask turns off the seventh bit in the target.

Target	**1 1 0 0 0 1 1 1**	***AND***
Mask	**1 0 1 1 1 1 1 1**	

Result	**1 0 0 0 0 1 1 1**	

Figure 4-15

Example of setting specific bits



Example 13

Use a mask to set the 5 leftmost bits of a pattern. Test the mask with the pattern 10100110.

Solution

The mask is 11111000.

Target	**1 0 1 0 0 1 1 0**	***OR***
Mask	**1 1 1 1 1 0 0 0**	

Result	**1 1 1 1 1 1 1 0**	

Example 14

Using the power plant example, how can you use a mask to to show that pump 6 is now turned on?

Solution

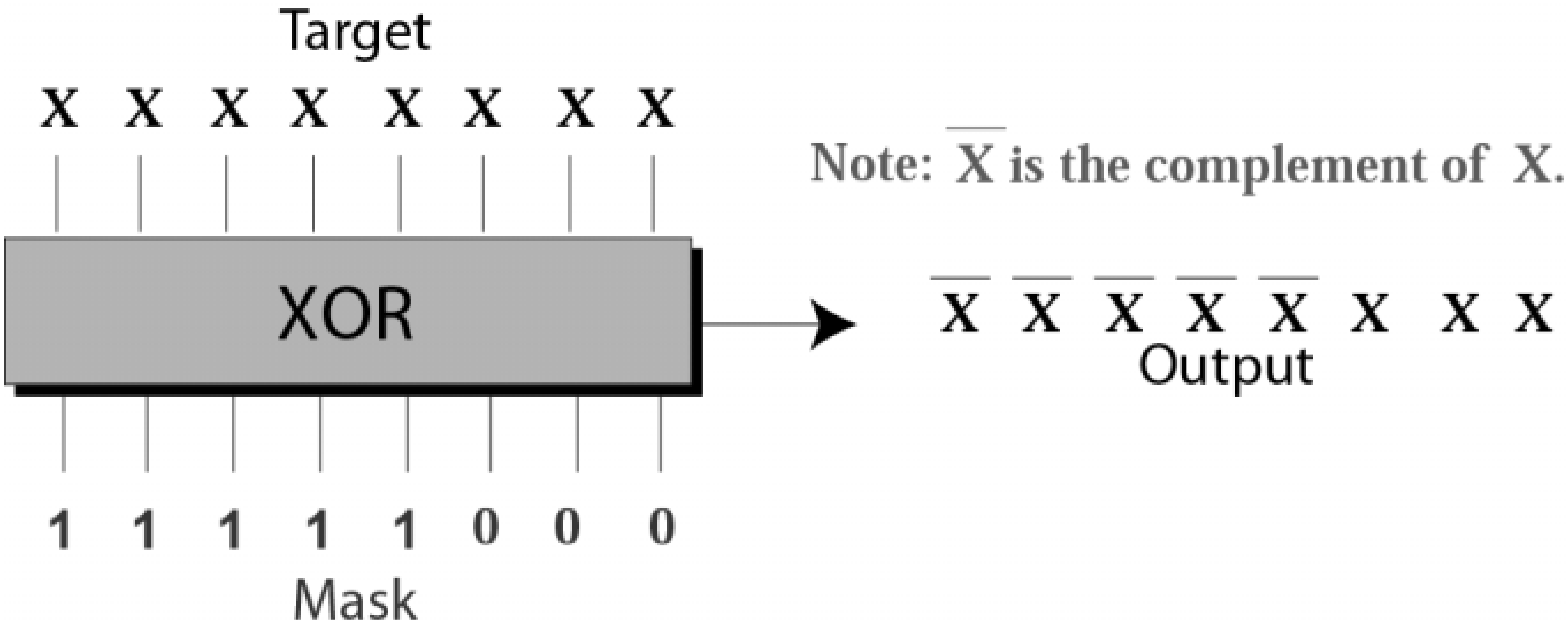
Use the mask 00100000.

Target	**1 0 0 0 0 1 1 1**	***OR***
Mask	**0 0 1 0 0 0 0 0**	

Result	**1 0 1 0 0 1 1 1**	

Figure 4-16

Example of flipping specific bits



Example 15

Use a mask to flip the 5 leftmost bits of a pattern. Test the mask with the pattern 10100110.

Solution

Target	**1 0 1 0 0 1 1 0**	***XOR***
Mask	**1 1 1 1 1 0 0 0**	

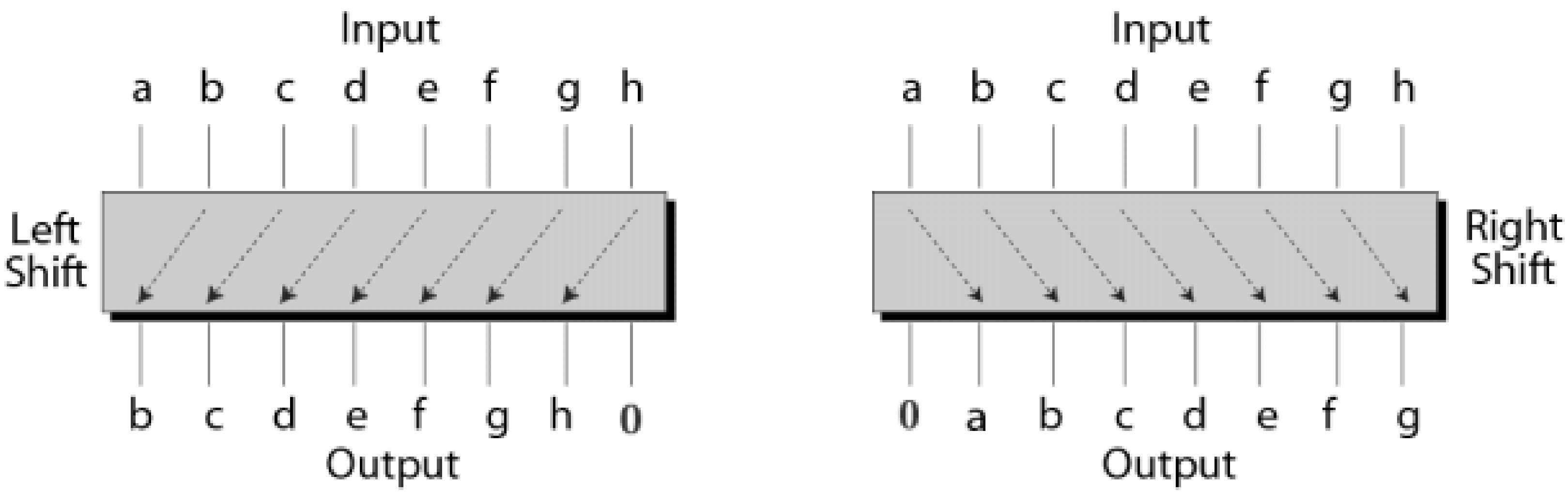
Result	**0 1 0 1 1 1 1 0**	

4.3

SHIFT OPERATIONS

Figure 4-17

Shift operations



Example 16

Show how you can divide or multiply a number by 2 using shift operations.

Solution

If a bit pattern represents an unsigned number, a right-shift operation divides the number by two. The pattern 00111011 represents 59. When you shift the number to the right, you get 00011101, which is 29. If you shift the original number to the left, you get 01110110, which is 118.

Example 17

Use a combination of logical and shift operations to find the value (0 or 1) of the fourth bit (from the right).

Solution

Use the mask 00001000 to AND with the target to keep the fourth bit and clear the rest of the bits.

Continued on the next slide

Solution (continued)

Target	**a b c d e f g h**	***AND***
Mask	**0 0 0 0 1 0 0 0**	

Result	**0 0 0 0 e 0 0 0**	

Shift the new pattern three times to the right

0000e000 ➔ 00000e00 ➔ 000000e0 ➔ 0000000e

Now it is easy to test the value of the new pattern as an unsigned integer. If the value is 1, the original bit was 1; otherwise the original bit was 0.