

Chapter 4



Operations on Bits

OBJECTIVES

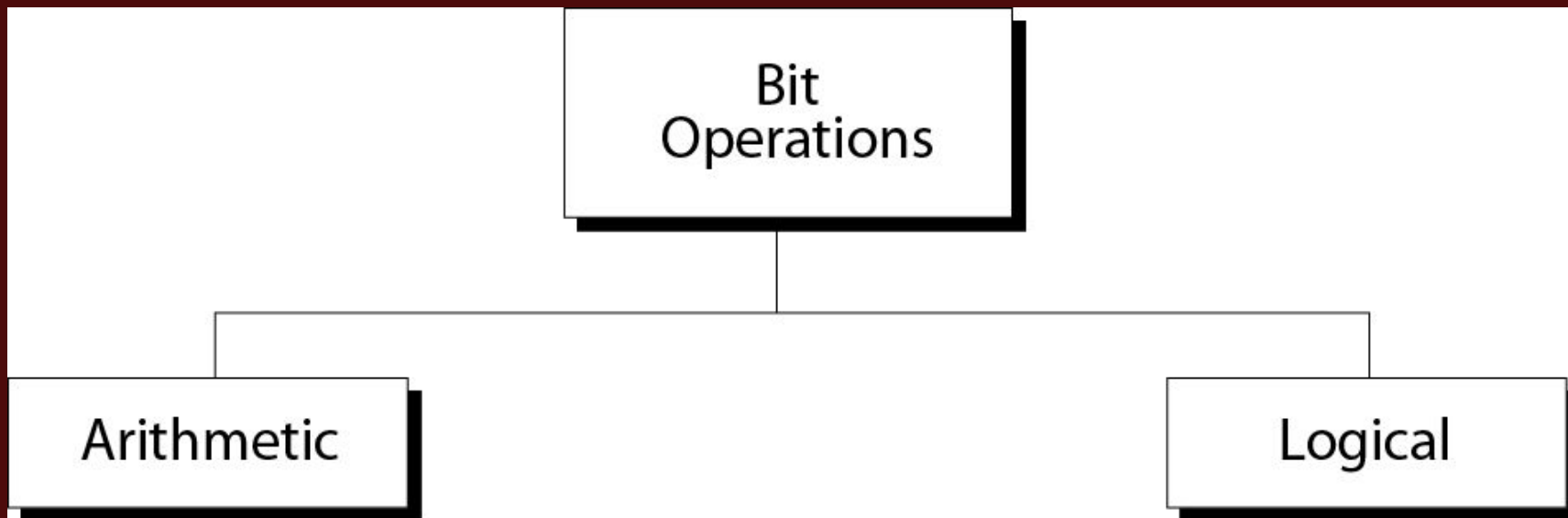


After reading this chapter, the reader should be able to:

- 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.



Operations on bits





4.1

ARITHMETIC OPERATIONS



Arithmetic operations

- Arithmetic operations involve:
 - Adding (+)
 - Subtracting (–)
 - Multiplying (X)
 - Dividing (/)
 - And so on...



Addition in two's complement

<i>Number of 1s</i>	<i>Result</i>	<i>Carry</i>
None	0	-
One	1	-
Two	0	1
Three	1	1

Table 4.1 Adding bits



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 (捨棄) it.**



Example 1

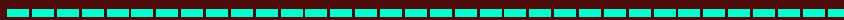
Add two numbers in two's complement representation: $(+17) + (+22) \square (+39)$

Solution

Carry

1

$$\begin{array}{rcccccccc} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & + \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{array}$$



Result 0 0 1 0 0 1 1 1 \square 39



Example 2

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

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 $\square +7$



Example 3

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

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 \square -15



Example 4

Add two numbers in two's complement representation: $(+127) + (+3) \square (+130)$

Solution

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

Result 1 0 0 0 0 0 1 0 \square *-126 (Error)*

An overflow has occurred.



Note:

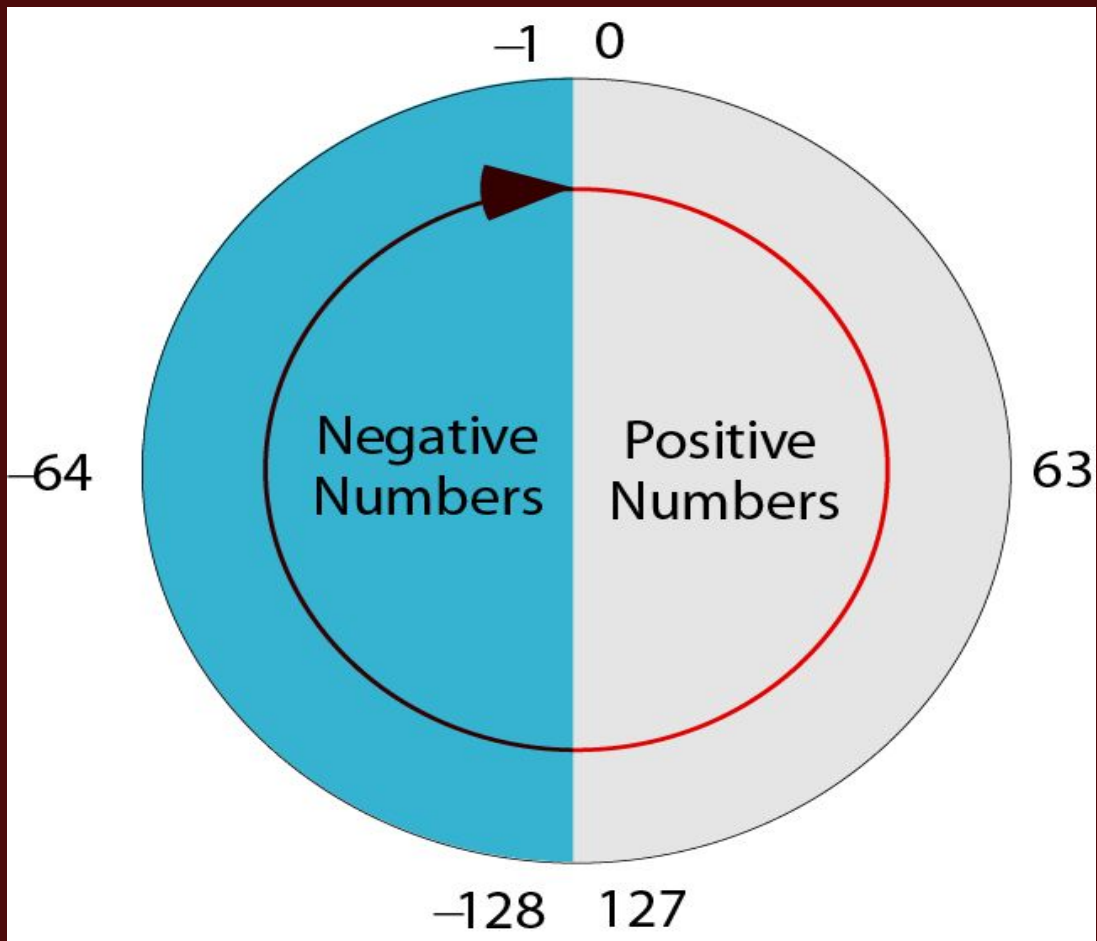
Range of numbers in two's complement representation

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

$-1)$



Two's complement numbers visualization





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.*

Subtraction in two's complement



Example 5

Subtract 62 from 101 in two's complement:

$$(+101) - (+62) \quad \square \square \quad (+101) + (-62)$$

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.

Arithmetic operations on floating-point numbers



- Addition and subtraction for floating-point numbers are one process. (p. 54)
 - Check the **sign**. (a, b)
 - Move the decimal points to make the **exponents** the **same**.
 - Add or subtract the **mantissas** (底數).
 - **Normalize** the result before storing in memory.
 - Check for any **overflow**.

Addition



Example 6

Add two floats:

0 10000100 1011000000000000000000000000

0 10000010 0110000000000000000000000000

Solution

The exponents are 5 and 3. The numbers are:

$+2^5 \times 1.1011$ and $+2^3 \times 1.011$

Make the exponents the same.

$(+2^5 \times 1.1011) + (+2^5 \times 0.01011) \square +2^5 \times 10.00001$

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

0 10000101 0000010000000000000000000000



4.2

LOGICAL OPERATIONS

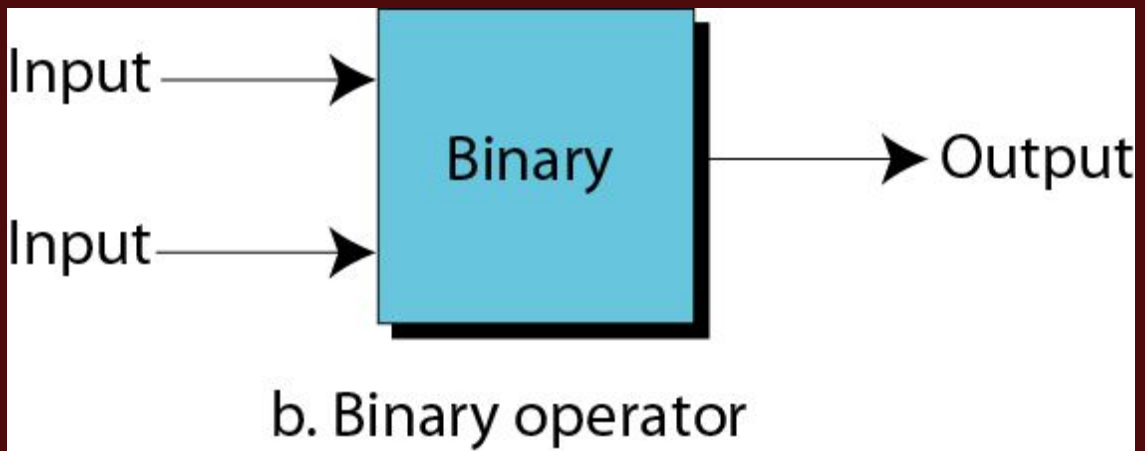
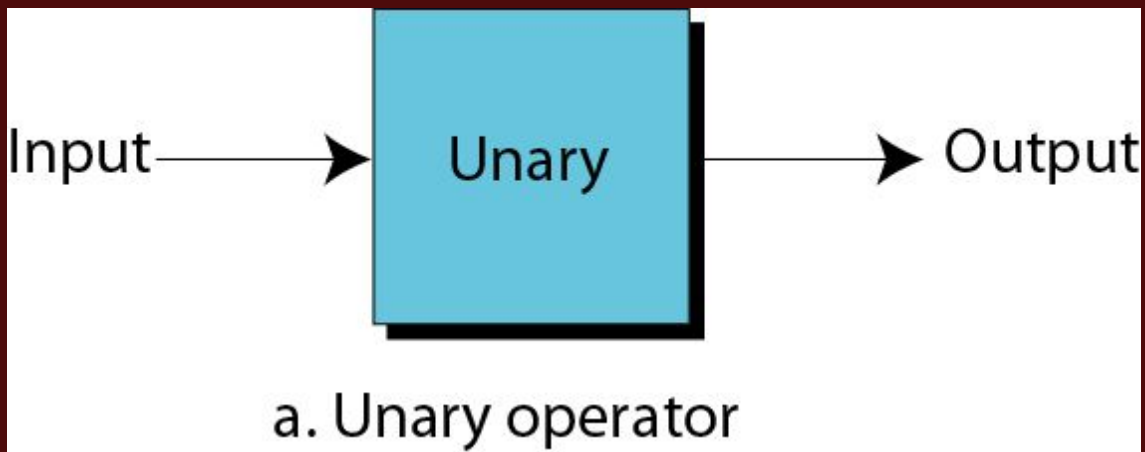


Logical operations

- A **logical operation** can accept **1** or **2** bits to create only **1** bit.
 - Unary operation (Figure4.3)
 - Binary operation (Figure4.3)

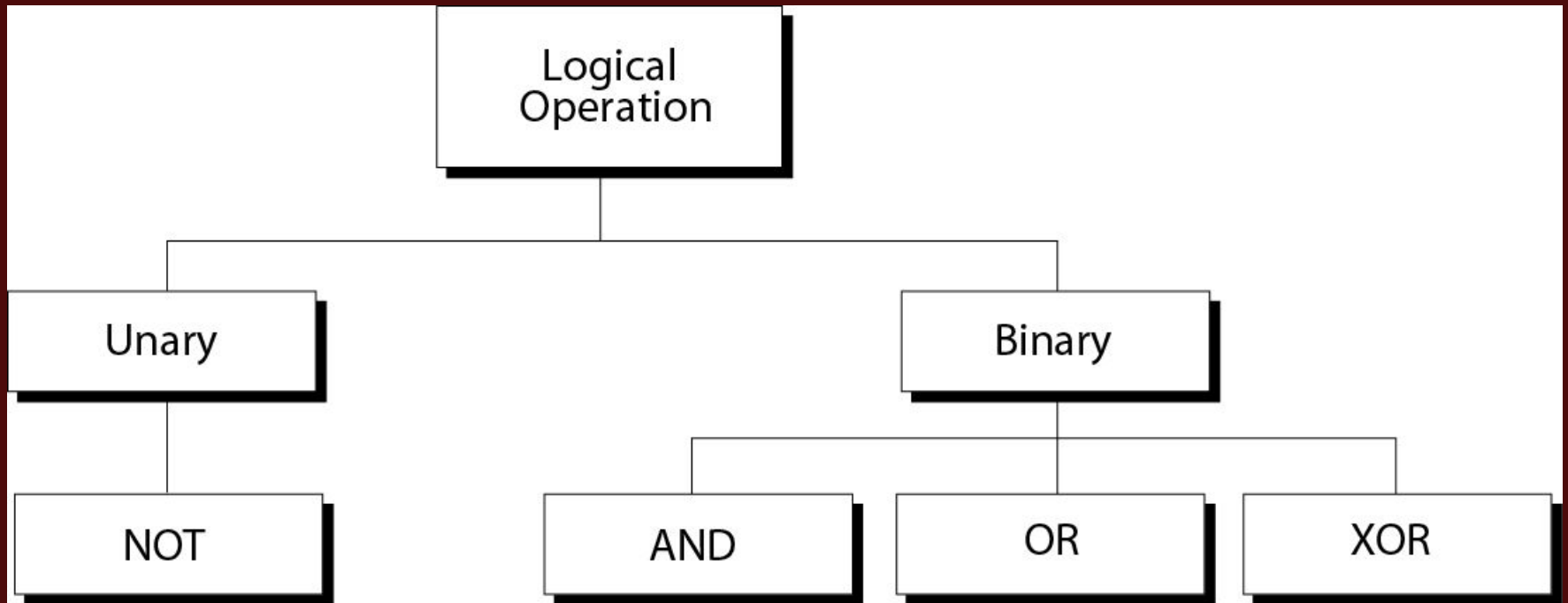


Unary and binary operations





Logical operations





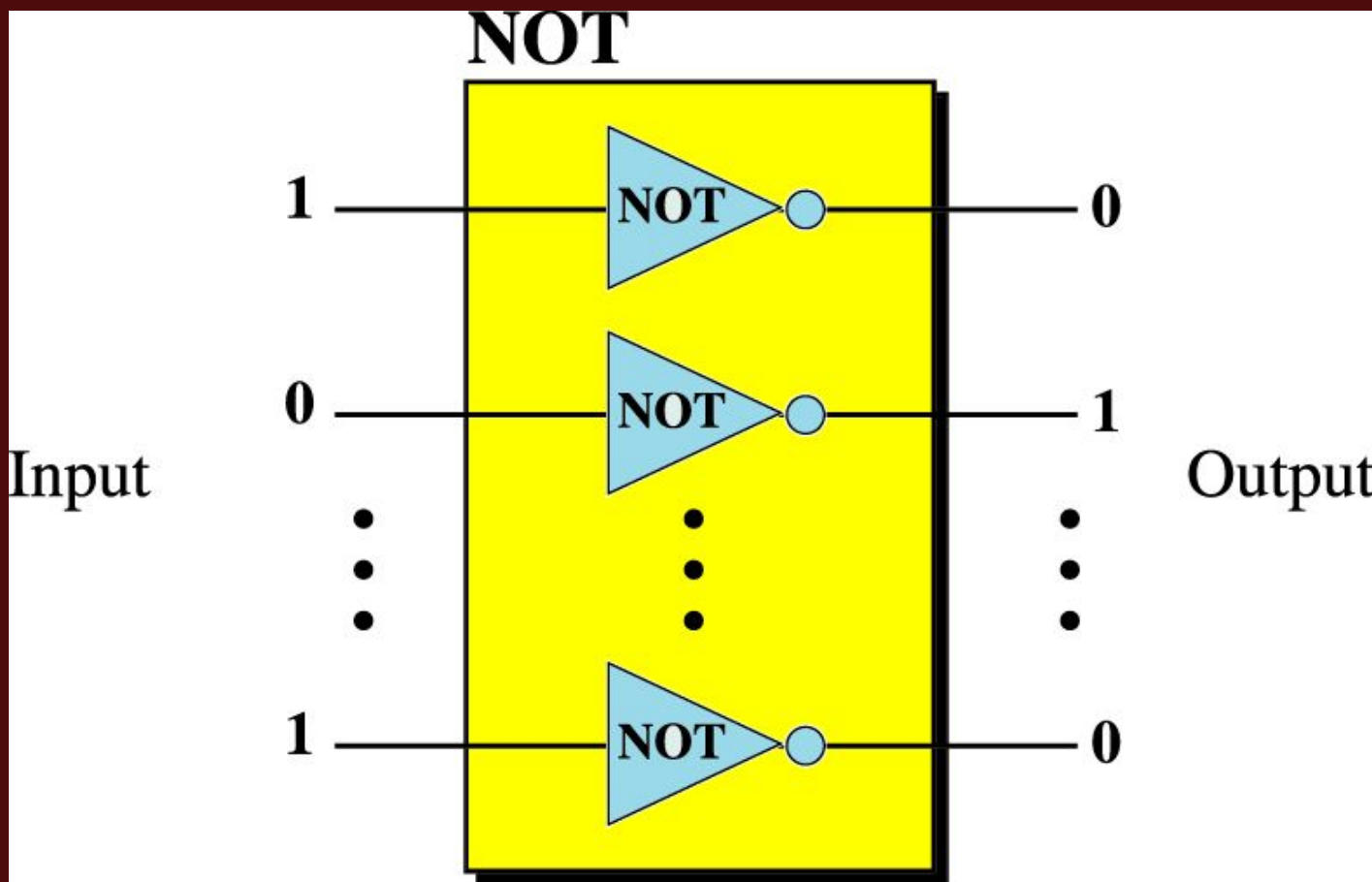
Truth tables

NOT		AND		
x	NOT x	x	y	x AND y
0	1	0	0	0
1	0	0	1	0
		1	0	0
		1	1	1

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



Unary operator -- NOT operator





NOT operator

Example 7

Use the NOT operator on the bit pattern 1001 1000

Solution

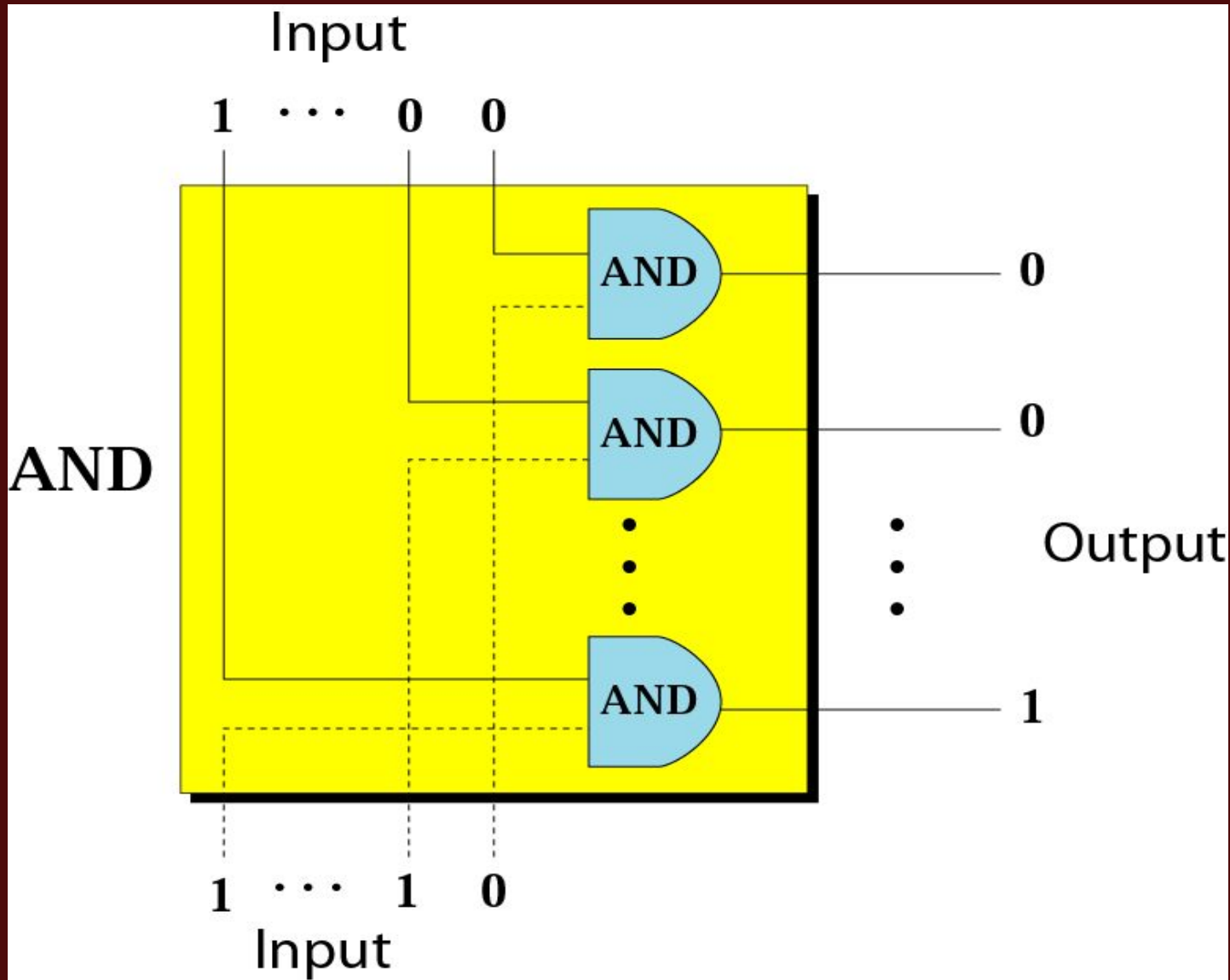
Target 1 0 0 1 1 0 0 0 *NOT*



Result 0 1 1 0 0 1 1 1

Figure 4-7

Binary operator--AND operator





AND operator

Example 8

Use the AND operator on bit patterns 10011000 and 00110101.

Solution

<i>Target</i>	1 0 0 1 1 0 0 0	<i>AND</i>
	0 0 1 1 0 1 0 1	

<i>Result</i>	0 0 0 1 0 0 0 0	



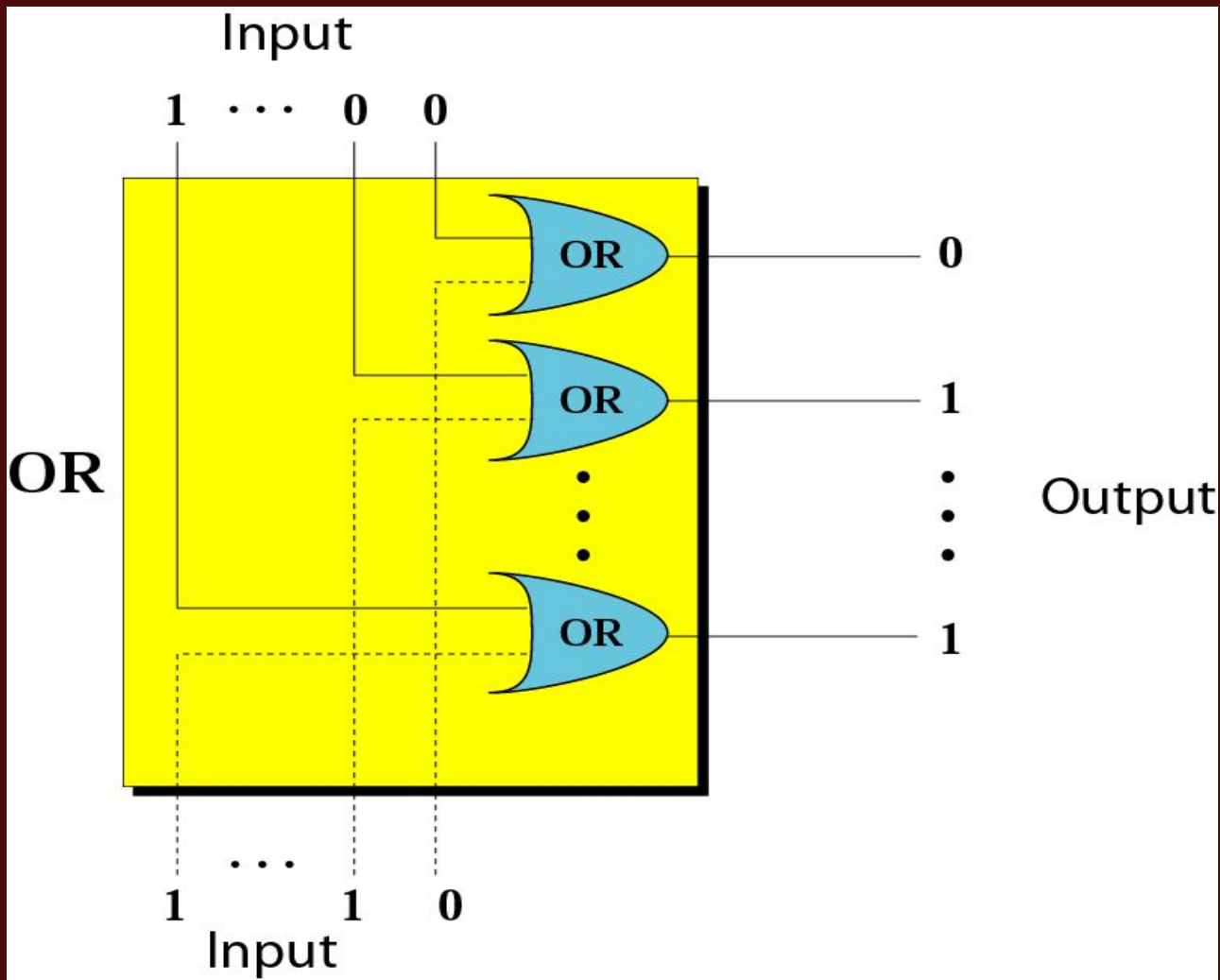
Inherent (本質的) rule of the AND operator

(0) AND (X) → (0)

(X) AND (0) → (0)



Binary operator--OR operator





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



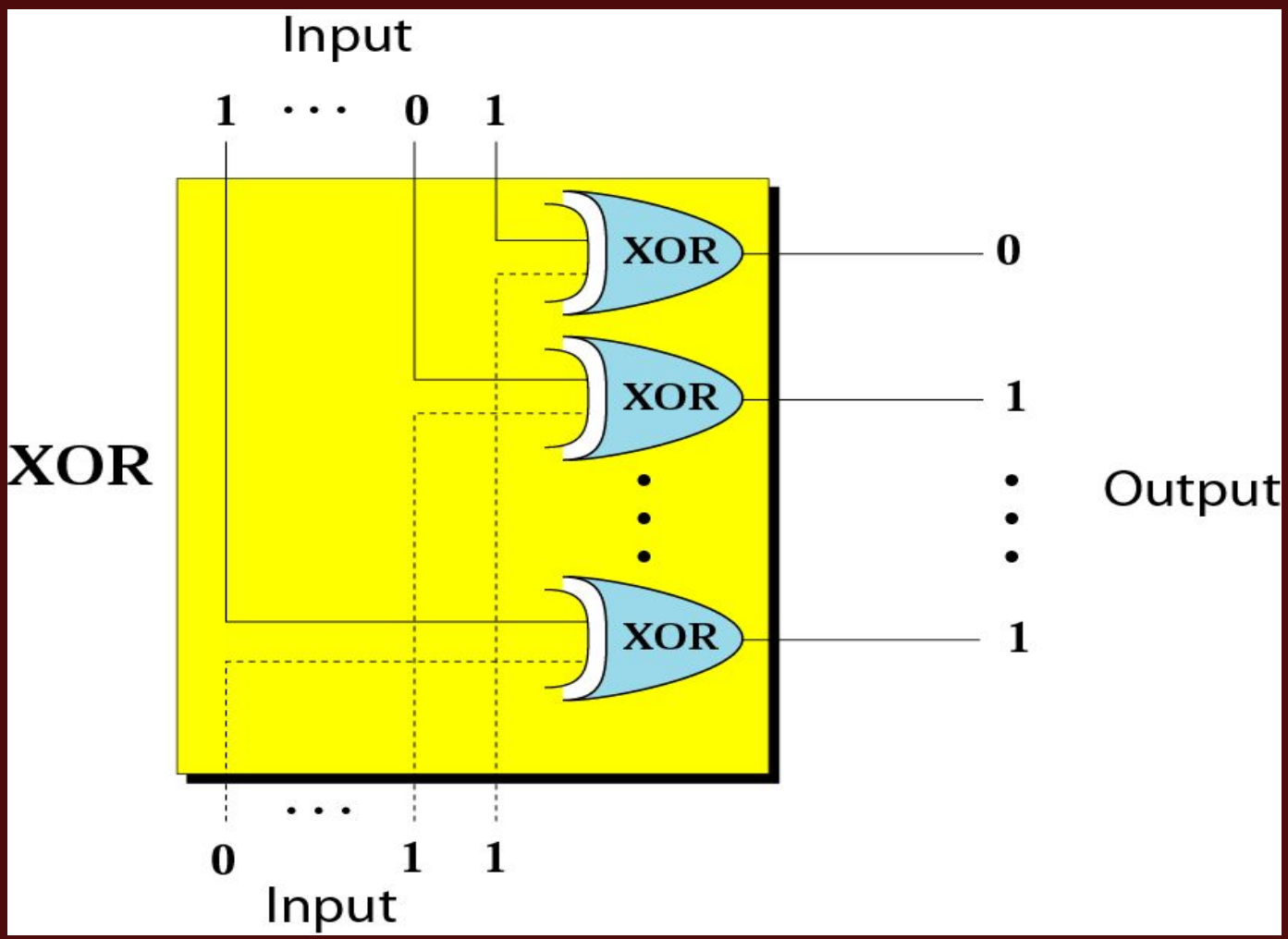
Inherent rule of the OR operator

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

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



Binary operator--XOR operator





XOR operator

Example 10

Use the XOR operator on bit patterns 10011000 and 00110101.

Solution

<i>Target</i>	1 0 0 1 1 0 0 0	<i>XOR</i>
	0 0 1 1 0 1 0 1	

<i>Result</i>	1 0 1 0 1 1 0 1	



Inherent rule of the XOR operator

(1) XOR (X) \longrightarrow NOT (X)

(X) XOR (1) \longrightarrow NOT (X)



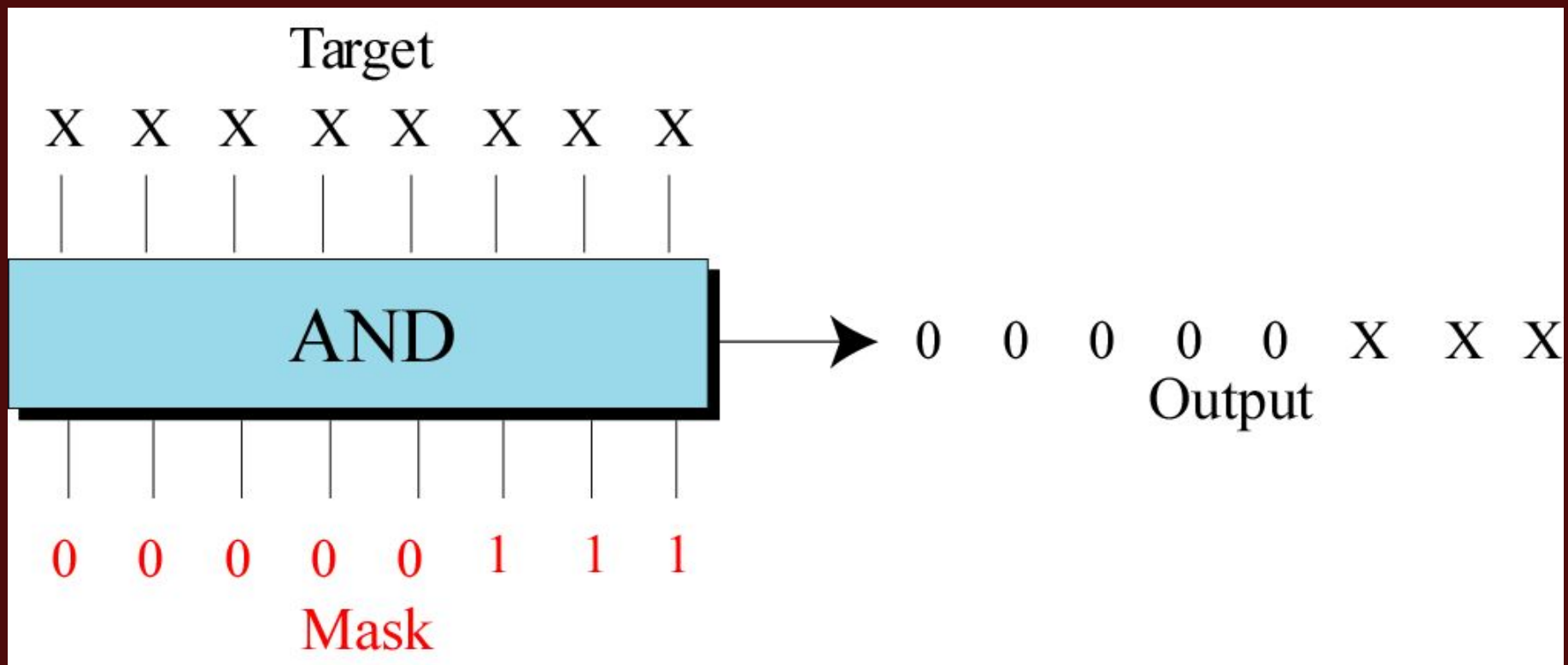
Applications

Mask (遮罩)





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** (水力發電廠) that **pumps water** (供水) to a city using **eight pumps** (抽水機). 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

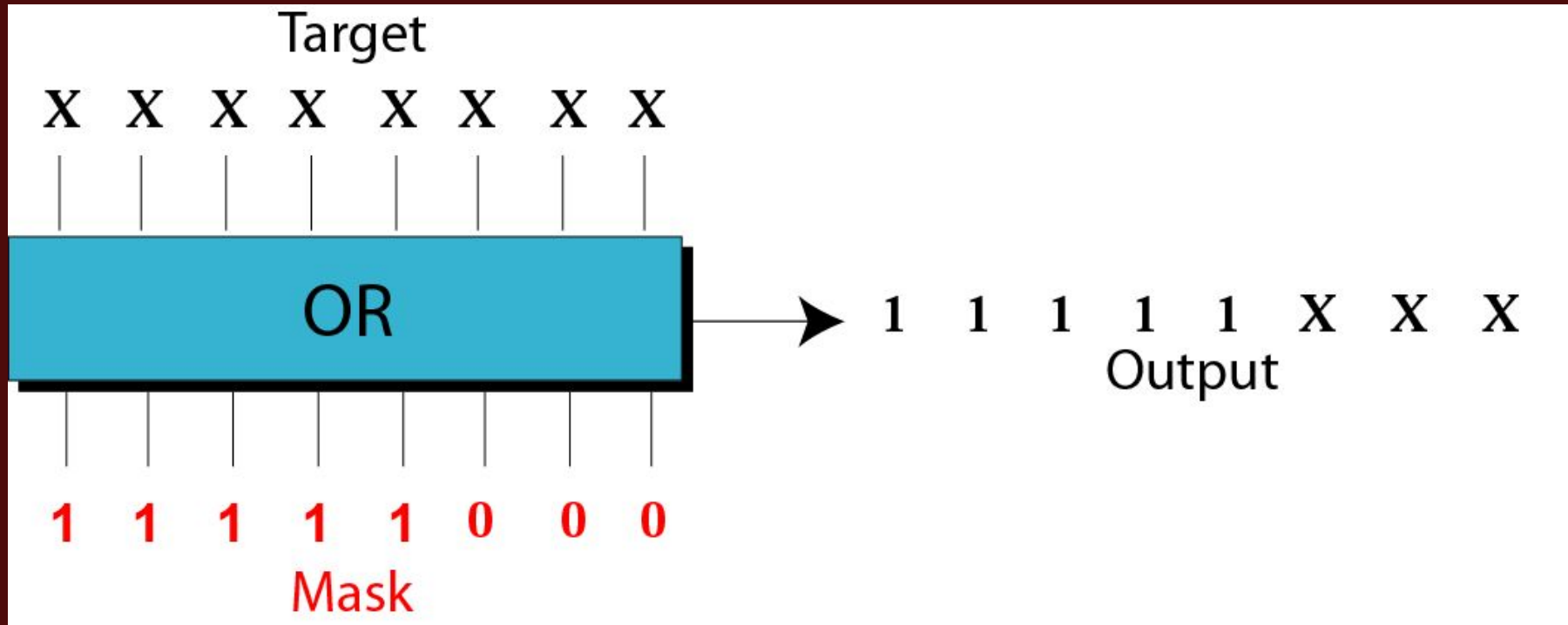
Use the mask 10111111 to AND with the target pattern. The only 0 bit (bit 7) in the mask turns off the seventh bit in the target.

<i>Target</i>	1 1 0 0 0 1 1 1	<i>AND</i>
<i>Mask</i>	1 0 1 1 1 1 1 1	

<i>Result</i>	1 0 0 0 0 1 1 1	



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 show that pump 6 is now turned on?

Solution

Use the mask **00100000**.

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

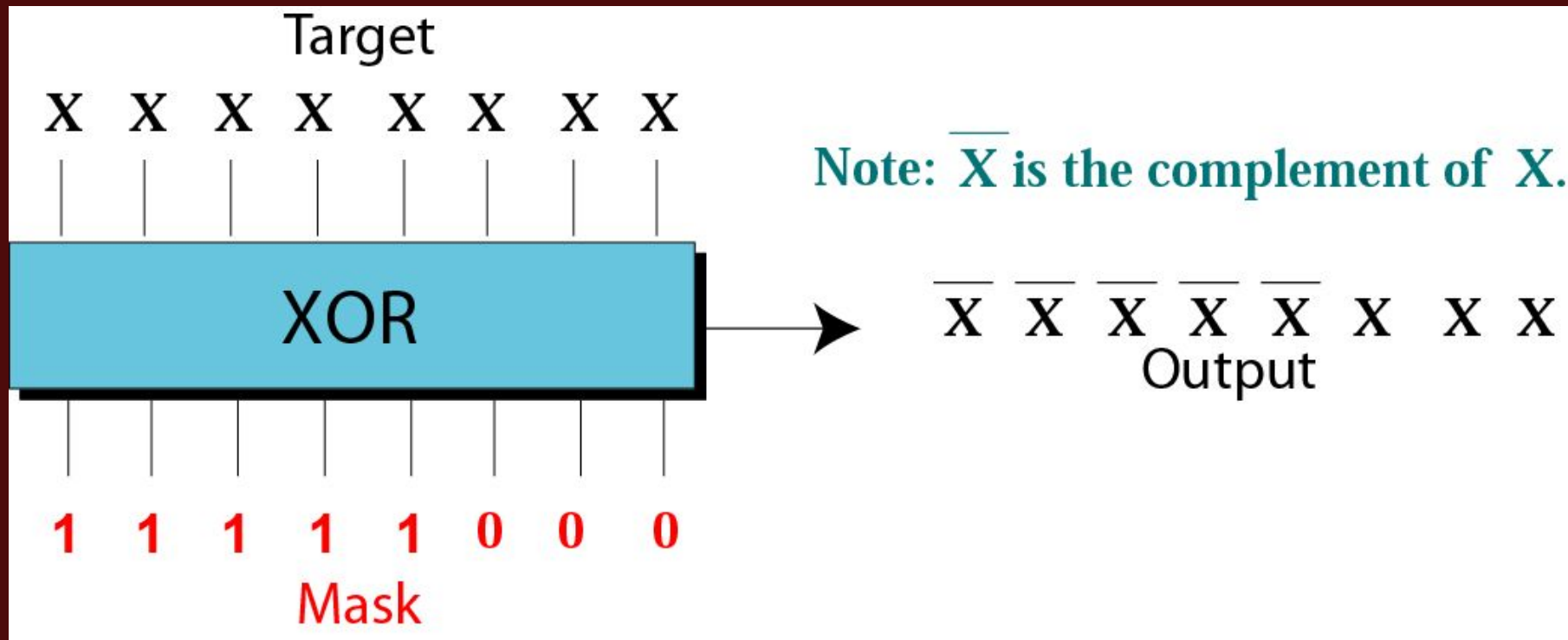
Mask 0 0 1 0 0 0 0 0



Result 1 0 1 0 0 1 1 1



Example of flipping (跳動的) 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



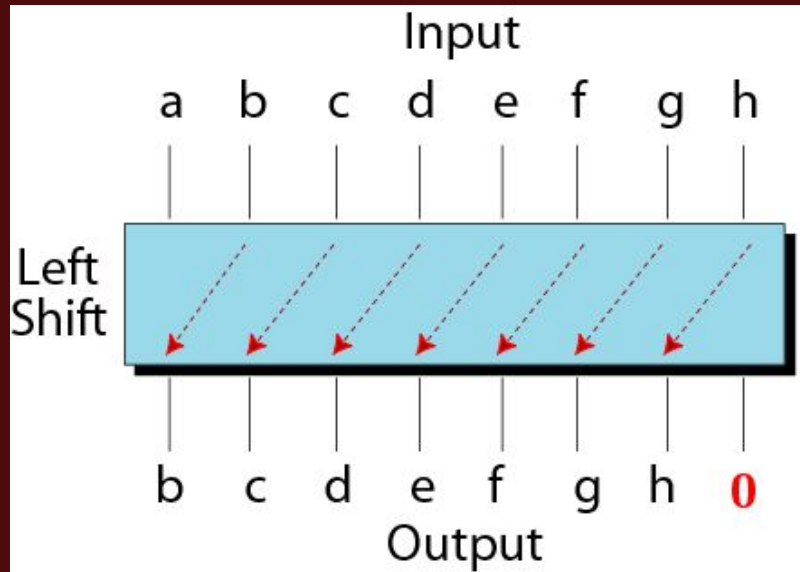
4.3

SHIFT OPERATIONS

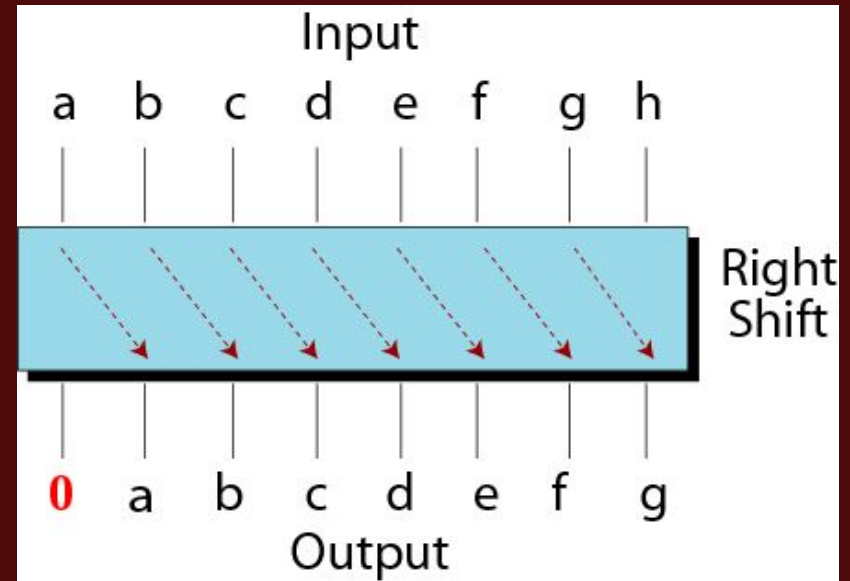


Shift operations

Left shift



Right shift





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.

Key terms



- AND operator
- Arithmetic operation
- Binary operation
- Binary operator
- Carry
- Clear
- Flip
- Floating-point number
- Force (強迫) to 0
- Force (強迫) to 1
- Logical operation
- Mantissa
- Mask
- NOT operator
- OR operator
- Overflow
- Set
- Truth table
- Two's complement
- Unary operation
- Unary operator
- Unset
- XOR operator