嫦娥

雲母屏風燭影深，
長河漸落曉星沉。
嫦娥應悔偷靈藥，
碧海青天夜夜心 。

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.

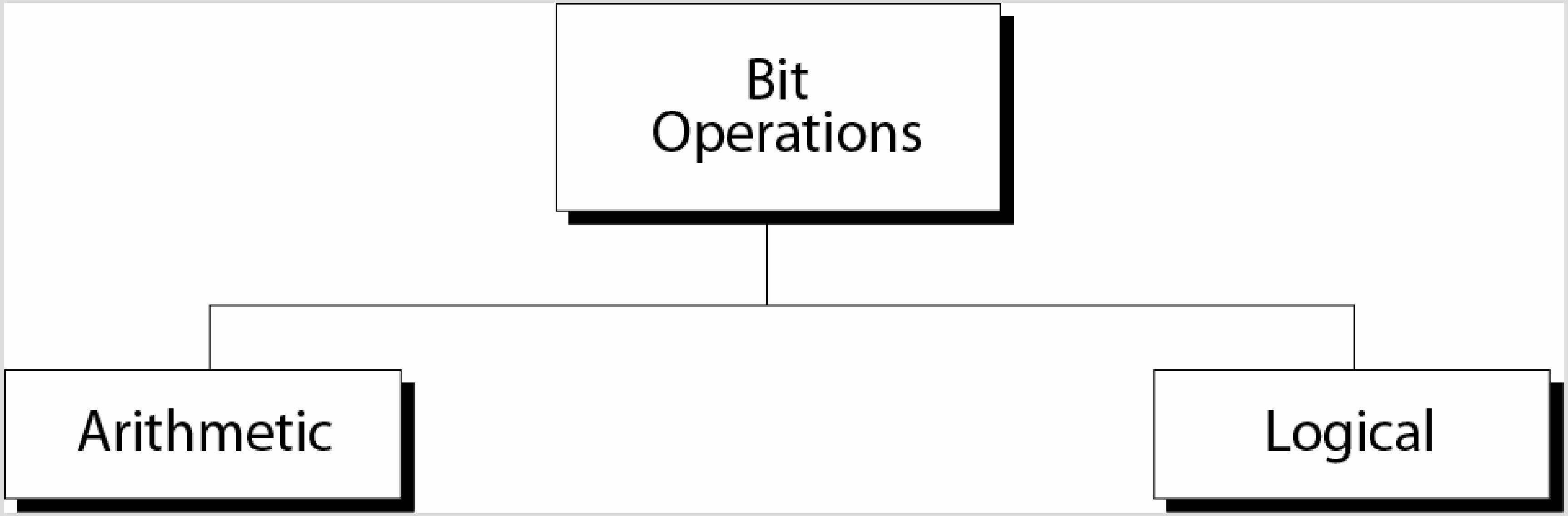
OBJECTIVES

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

You can perform arithmetic or logical operations on bits



4.1

ARITHMETIC OPERATIONS

Adding bits

- Most computers use the two's complement method of integer representation.
- Adding numbers in two's complement is like adding the numbers in decimal, if there is a carry, it is added to the next column
- If there is a carry after addition of the leftmost digits, the carry is discarded

Number of 1s	***Result***	***Carry***
None	0	
One	1	
Two	0	1
Three	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

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.

Overflow

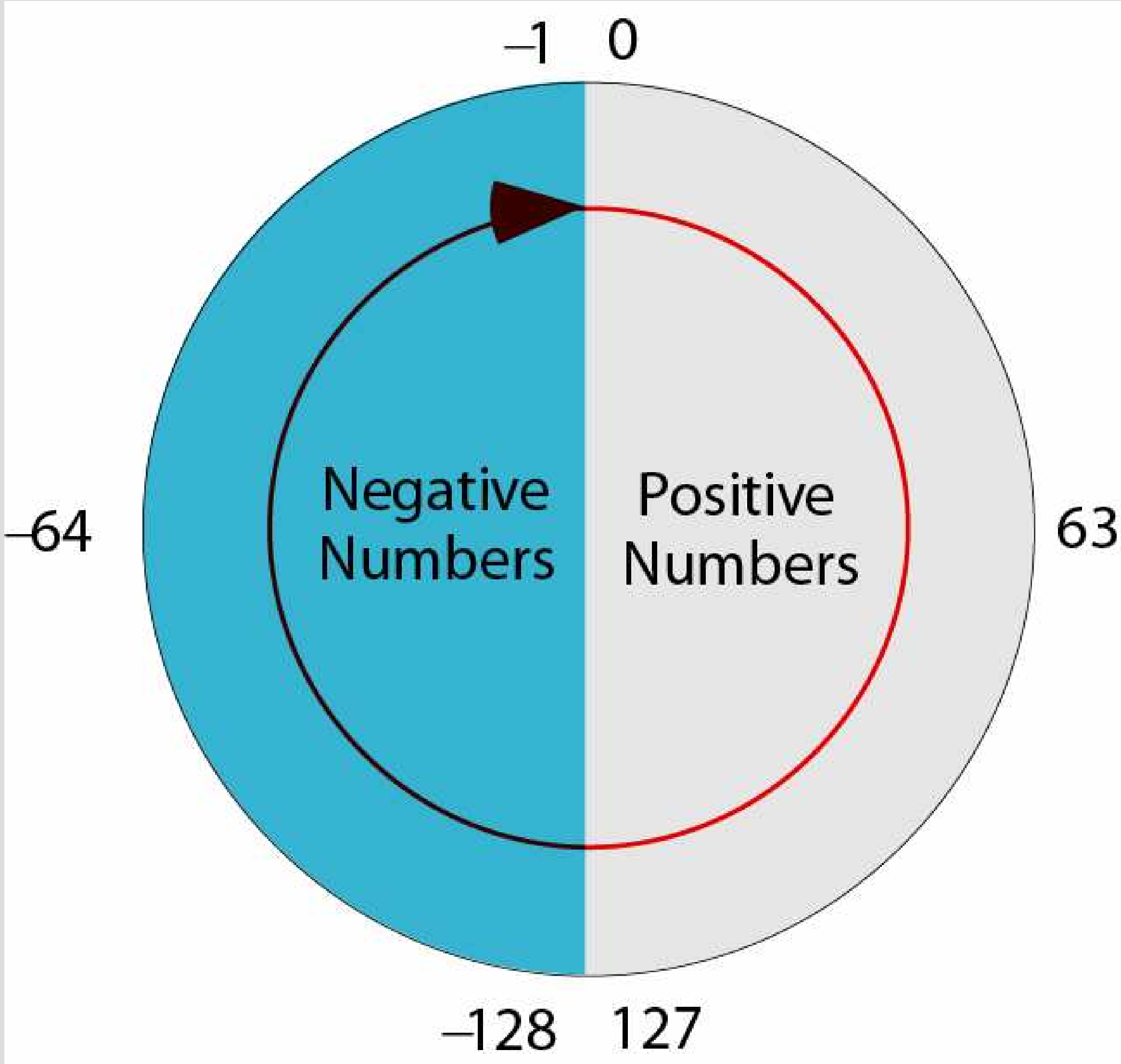
- The term overflow describes a condition in which a number is not within the range defined by the bit allocation
- For example4, the range is -2^{8-1} to $+2^{8-1}-1$, which is -128 to 127. The result of the addition (130) is not in this range

Note:

Range of numbers in two's complement representation

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

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

- To subtract in two's complement, just negate the number to be subtracted and add

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.

Arithmetic on floating-point number

- The steps are as follows:
  - Check the signs
    - If the signs are the same, add the numbers and assign the sign to the result
    - If the signs are different, compare the absolute values, subtract the smaller from the larger, and use the sign of the larger for the result
  - Move the decimal points to make the exponents the same.
  - Add or subtract the mantissas
  - Normalize the result before storing in memory
  - Check for any overflow

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:

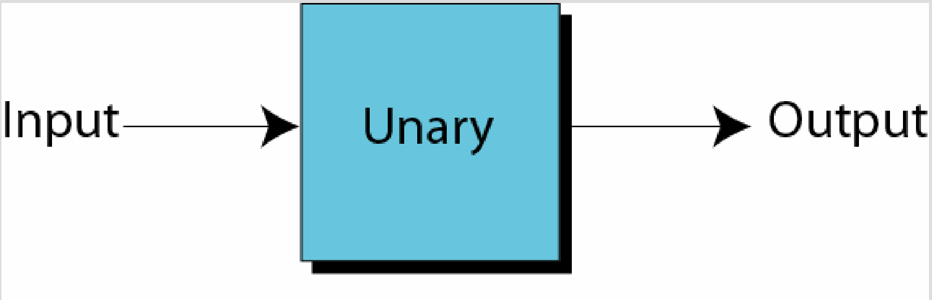
0 10000101 00000100000000000000000

4.2

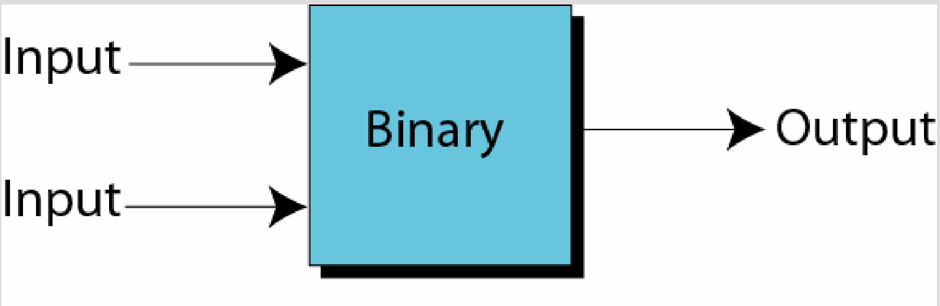
LOGICAL OPERATIONS

Unary and binary operations

- Logical operation on bits can be unary (one input) or binary (two inputs)

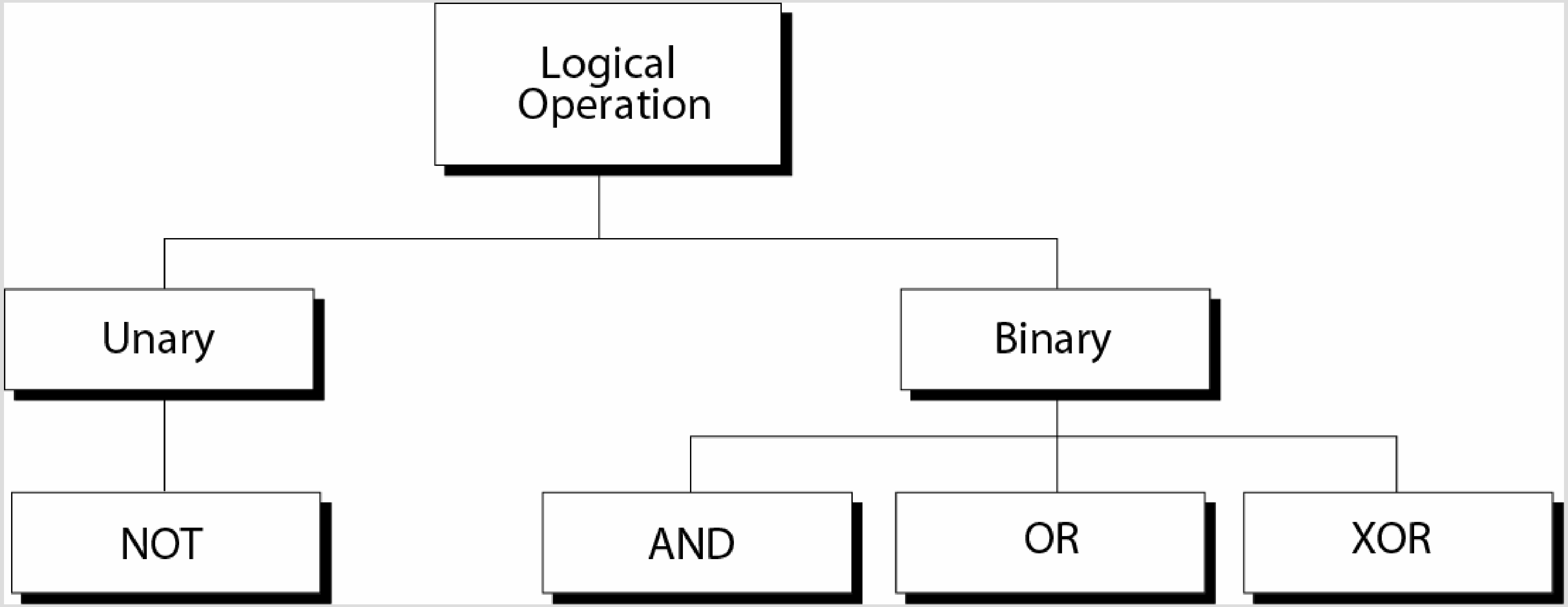


a. Unary operator



b. Binary operator

Logical operations



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

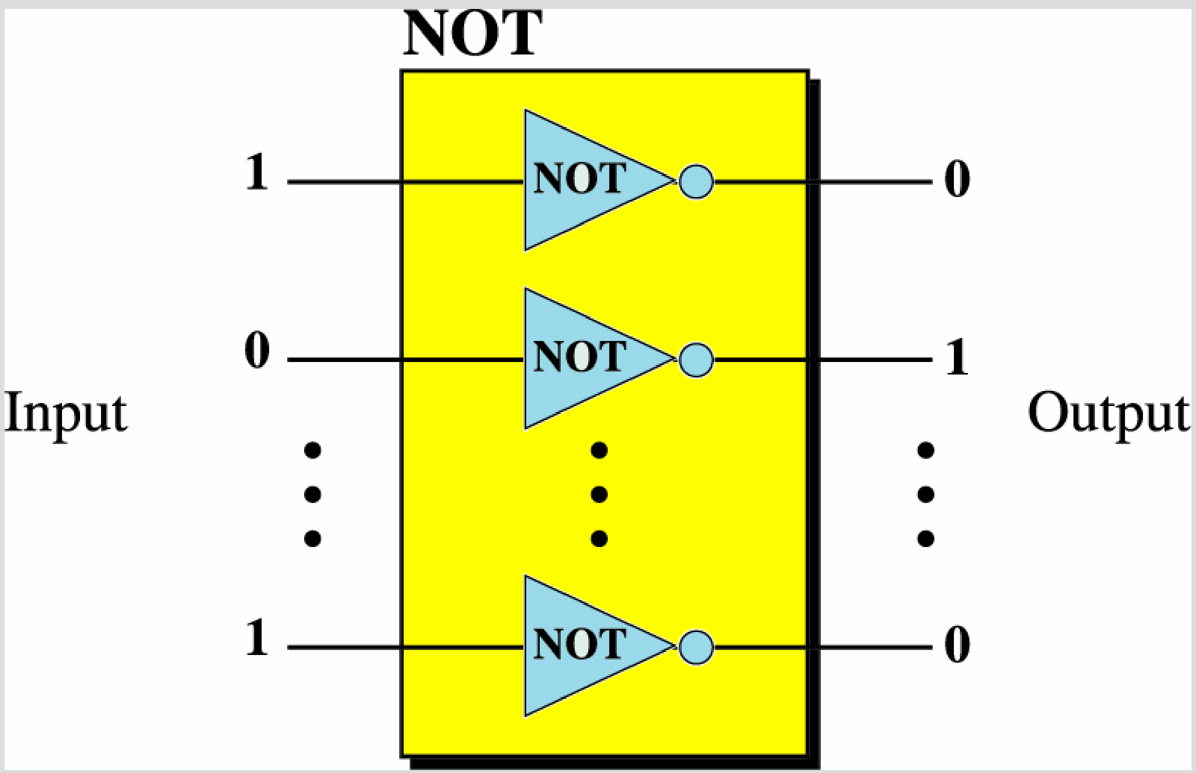
x	y	x OR y
0	0	0
0	1	1
1	0	1
1	1	1

XOR

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

NOT operator

The unary NOT operator inverts its input.



Example 7

Use the NOT operator on the bit pattern 10011000

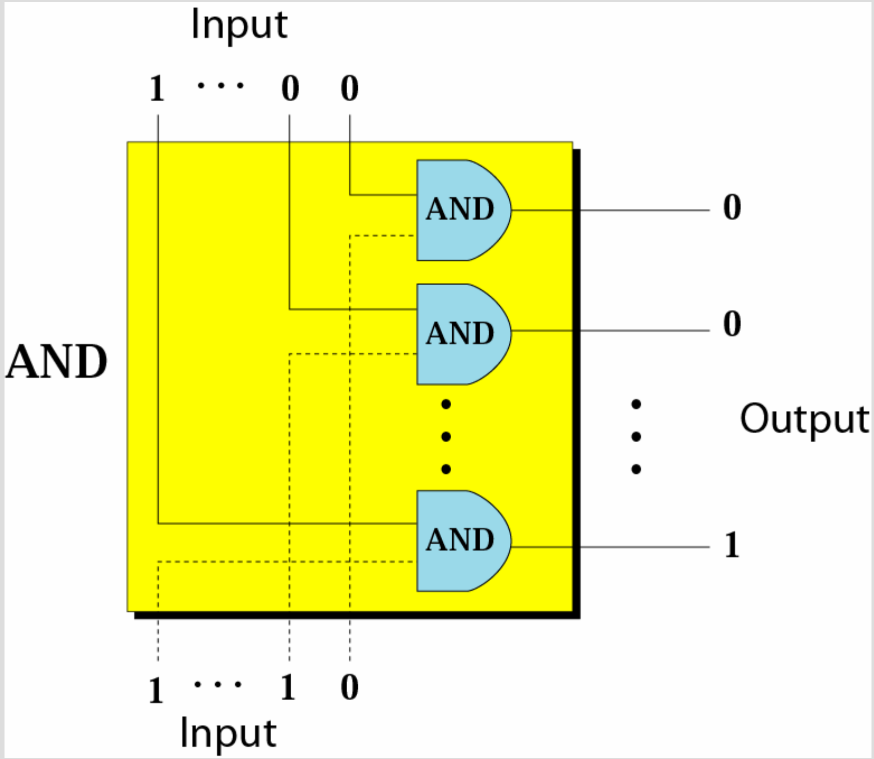
Solution

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

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

AND operator

The result of the binary AND operation is true only if both inputs are true.



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**	

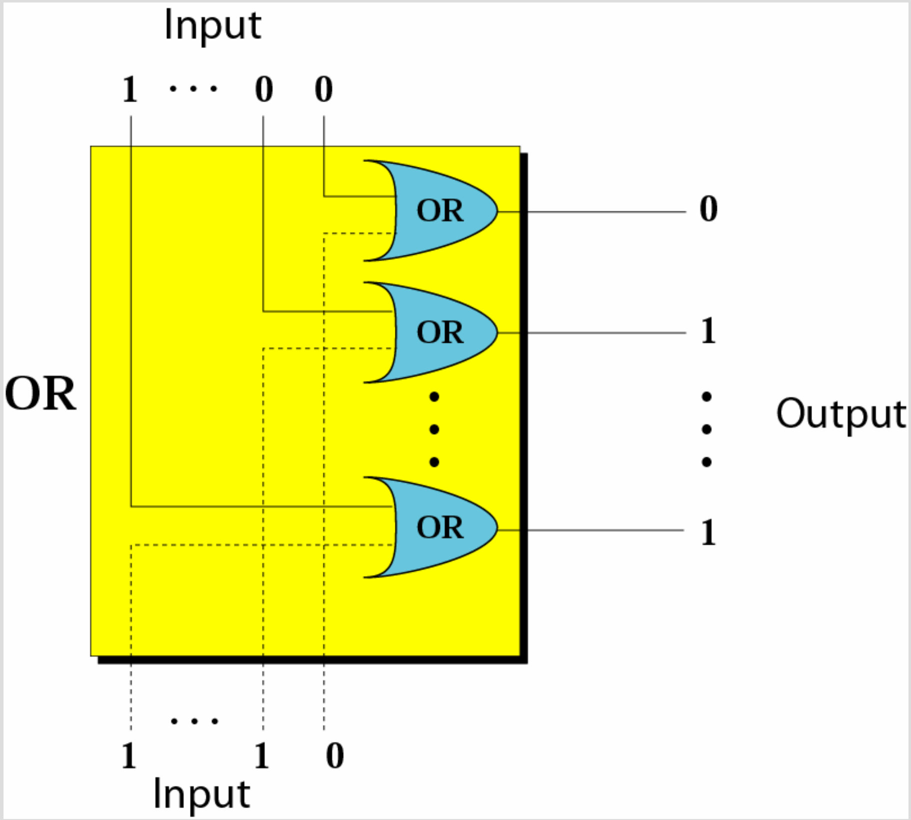
Inherent rule of the AND operator

If a bit in one input is 0, you do not have to check the corresponding bit in the other input $\Rightarrow$ the result is 0

(0)	**AND**	**(X)**	$\longrightarrow$	**(0)**
(X)	**AND**	**(0)**	$\longrightarrow$	**(0)**

OR operator

The result of the binary OR operation is false only if both inputs are false.



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**	

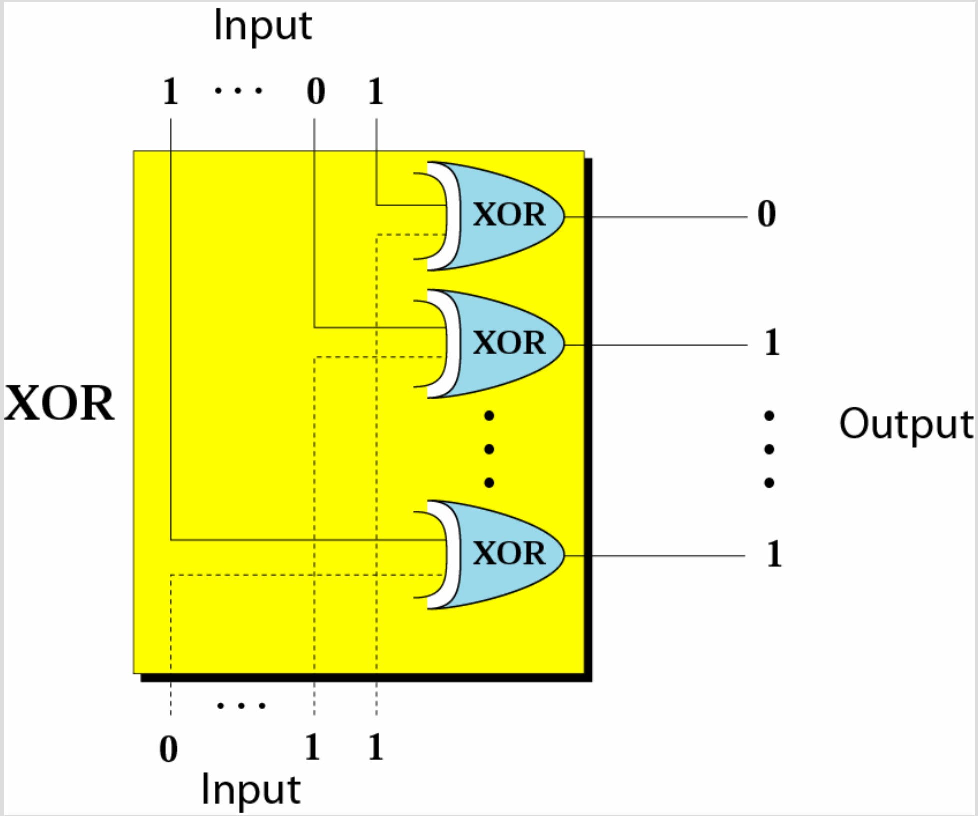
Inherent rule of the OR operator

If a bit in one input is 1, you do not have to check the corresponding bit in the other input $\Rightarrow$ the result is 1

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

XOR operator

The result of the binary XOR operation is false only if both input are the same.



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**	

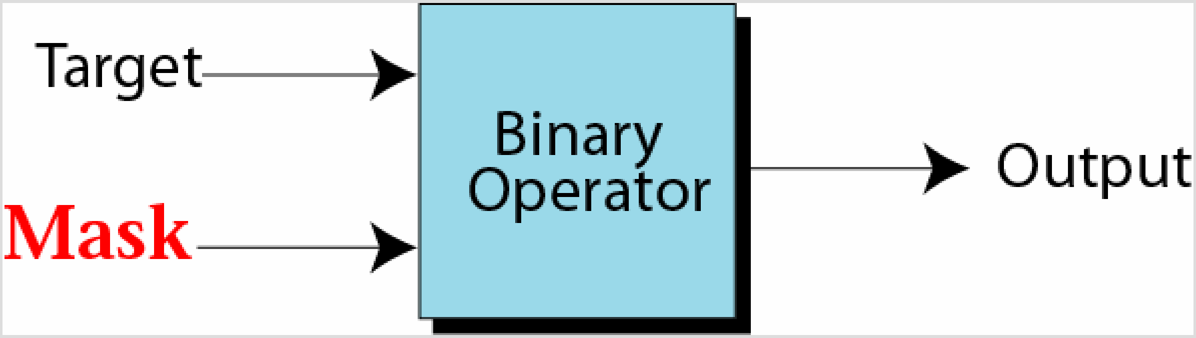
Inherent rule of the XOR operator

If a bit in one input is 1, the result is the inverse of the corresponding bit in the other input.

(1) XOR (X) → NOT (X)

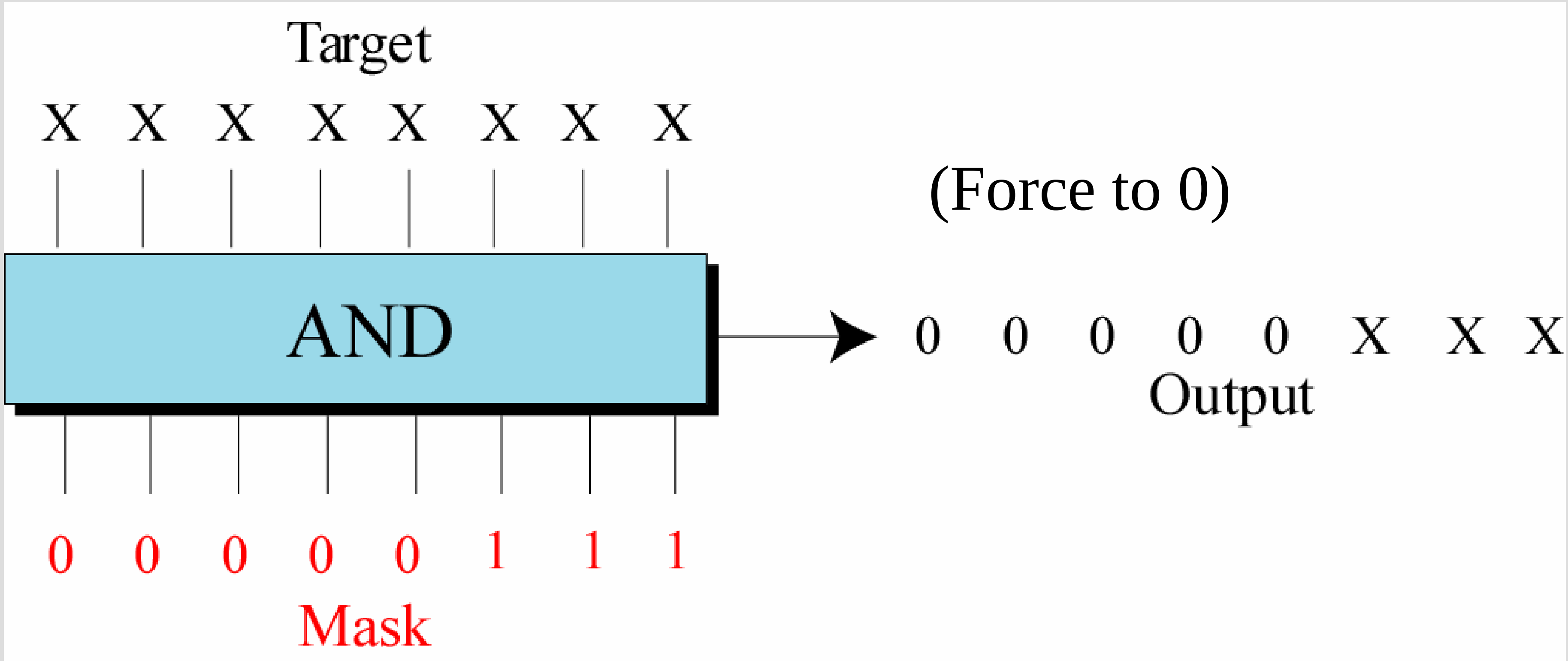
(X) XOR (1) → NOT (X)

Mask



A mask is a bit pattern that is applied to a target bit pattern to achieve a specific result.

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.

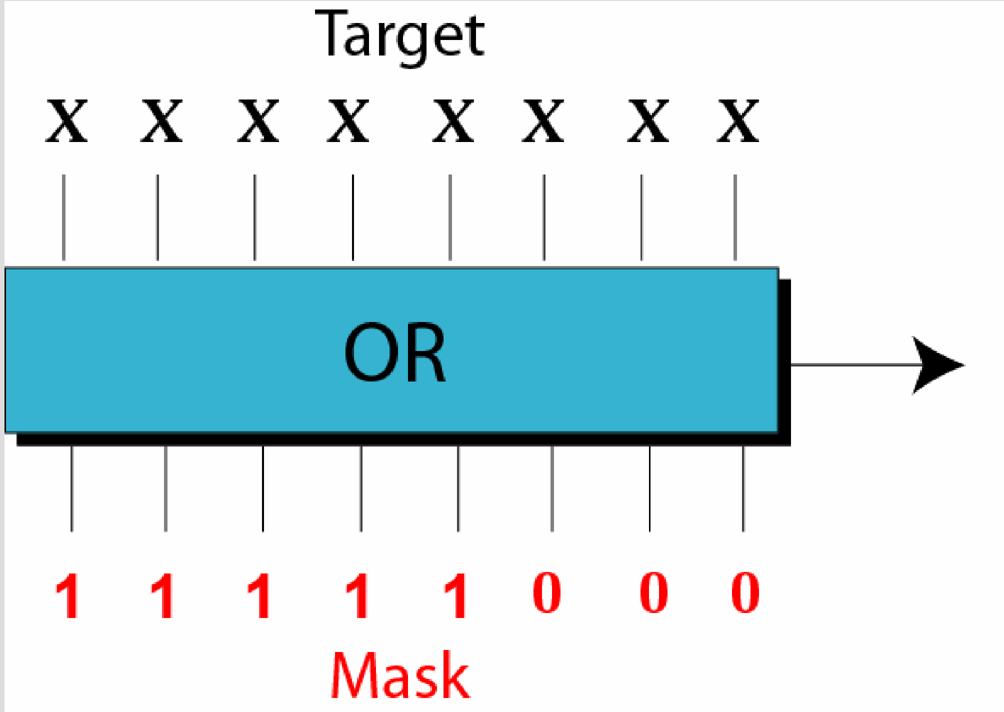
Solution

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**	

Setting specific bits



(Force to 1)

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

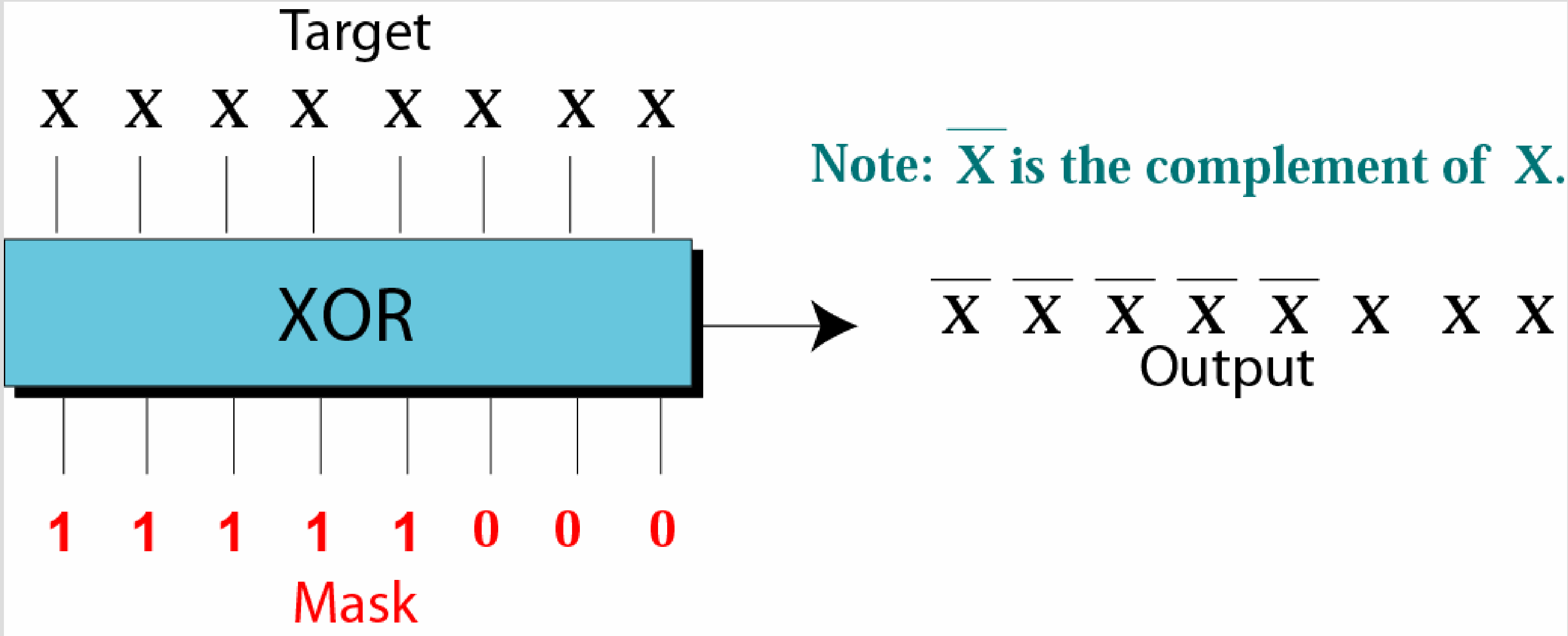
Use the mask 00100000.

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**	

Flipping specific bits

To change the value of specific bits from 0s to 1s, and vice versa



Example 15

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

Solution

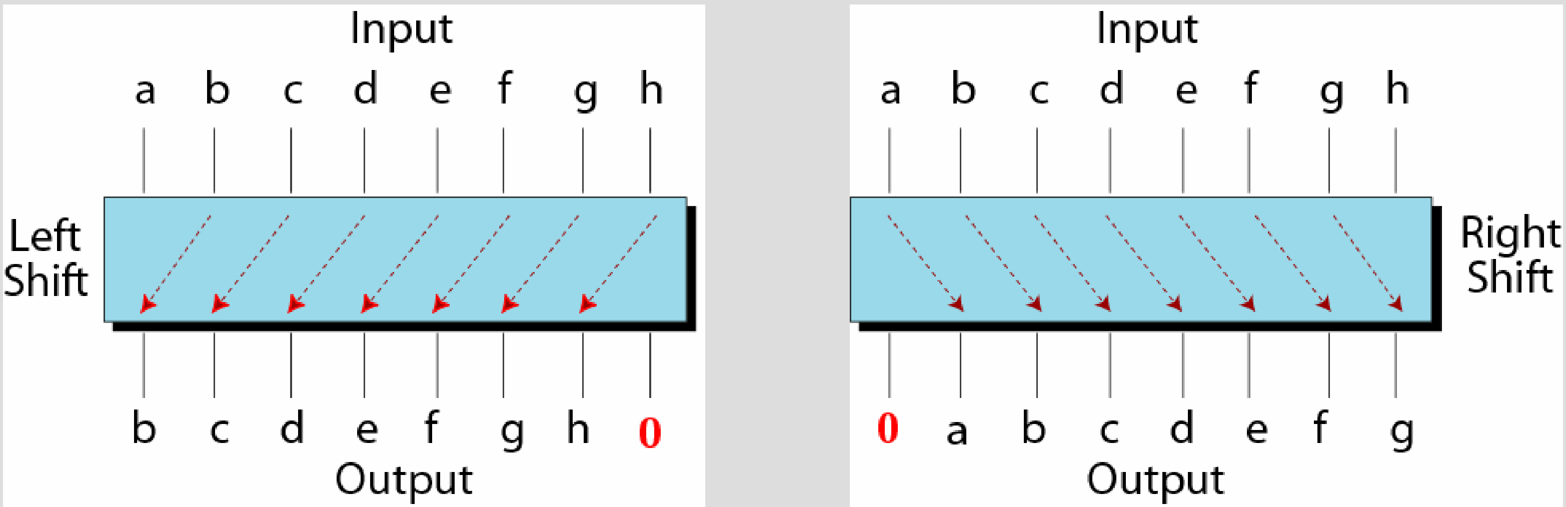
Target	**1 0 1 0 0 1 1 0**	***XOR***
Mask	**1 1 1 1 1 0 0 0**	

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

4.3

SHIFT OPERATIONS

Shift operations



- A bit pattern can be shifted to the right or to the left.
- The right-shift operation discards the rightmost bit, shifts every bit to the right, and inserts 0 as the leftmost bit.

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.