Chapter 4

# *Operations on Bits*

## Two's Complement :: Review

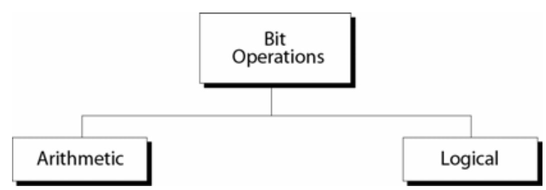
***In two's complement representation, the leftmost bit defines the sign of the number. If it is 0, the number is positive. If it is 1, the number is negative.***

## Two's Complement :: Review

***Two's complement can be achieved by reversing all bits except the rightmost bits up to the first 1 (inclusive). If you two's complement a positive number, you get the corresponding negative number. If you two's complement a negative number, you get the corresponding positive number. If you two's complement a number twice, you get the original number.***

Figure 4-1

## Operations on bits :: Review




## OBJECTIVES

- 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.

## Reading

- Chapter 4

**4.2**

# LOGICAL OPERATIONS

## Logical Operations

- A single bit can be either 0 or 1
- We can interpret 0 and 1 logically
- 0 – false;
- 1 – true;
- Logical operation – operations applied on bits interpreted as logical values

Figure 4-3

## Unary and binary operations

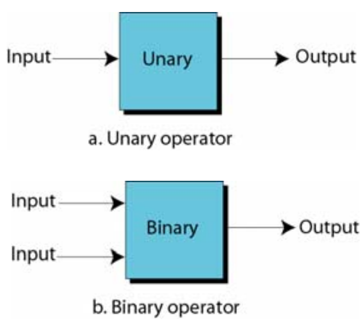



Figure 4-4

## Logical operations

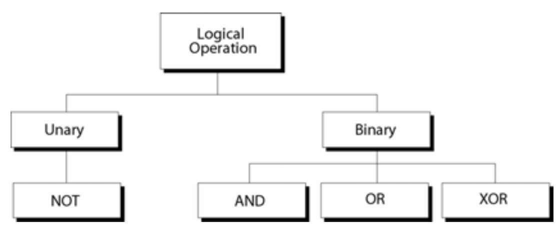



Figure 4-5

## 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

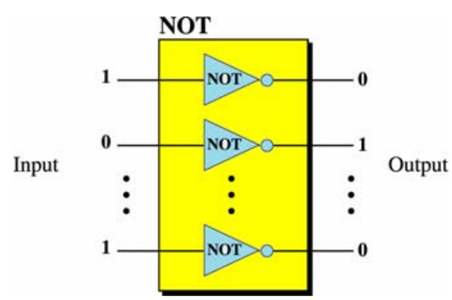
| x | y | x OR y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

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*

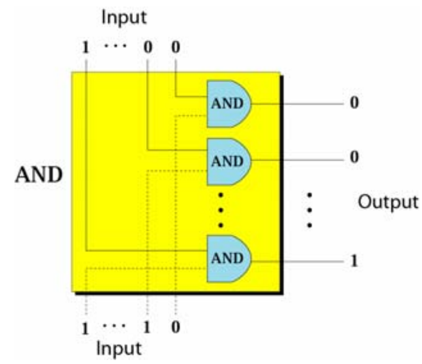
Use the NOT operator on the bit pattern 10011000

**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

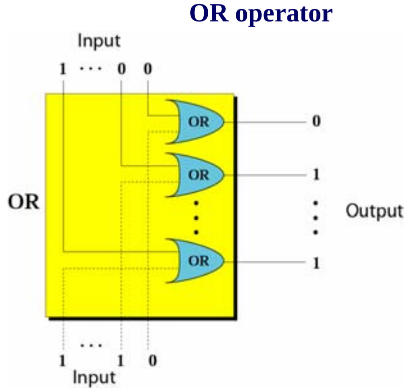
**Inherent rule of the AND operator**

(0) AND (X) → (0)

(X) AND (0) → (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

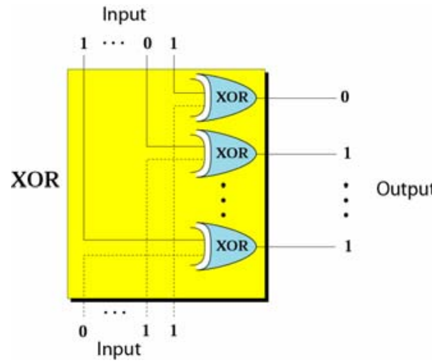
## Inherent rule of the OR operator

(1) OR (X) → (1)

(X) OR (1) → (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) XOR (X) → NOT (X)

(X) XOR (1) → NOT (X)

Figure 4-13

## Mask

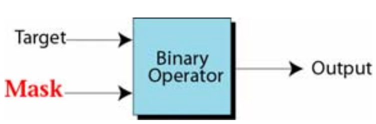
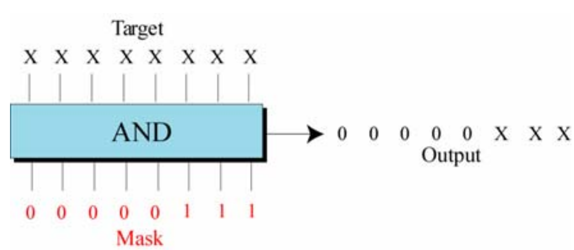



Figure 4-14

## 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.***

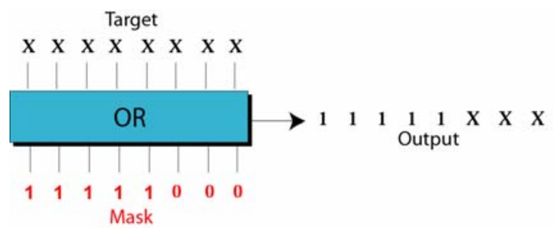
**Solution**

***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?

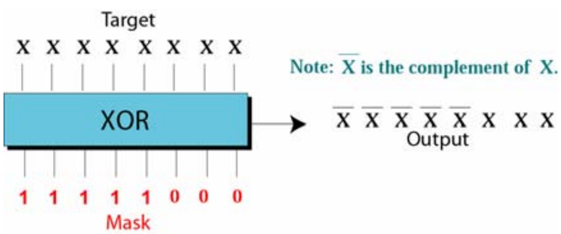
**Solution**

***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***

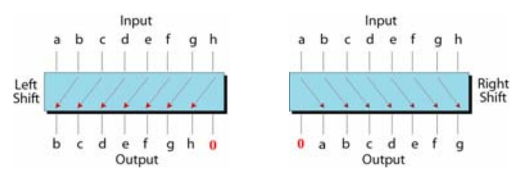
| | | |
|---|---|---|
| ***Target*** | **1 0 1 0 0 1 1 0** | ***XOR*** |
| ***Mask*** | **1 1 1 1 1 0 0 0** | |
| | ------------------ | |
| ***Result*** | **0 1 0 1 1 1 1 0** | |

***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.**

### Summary

- You can perform arithmetic and logical operations on bits
- Logical operations on bits can be unary or binary
- The unary NOT operator inverts its input
- The result of the binary AND operation is true only if both inputs are true

### Summary

- The result of the binary OR operation is false only if both inputs are false
- The result of the binary XOR operation is false only if both inputs are the same
- A mask is a bit pattern that is applied to a target bit pattern to achieve a specific result
- To clear a bit in a target bit pattern, set the corresponding mask bit to 0 and use AND

### Summary

- To set a bit in a target bit pattern, set the corresponding mask bit to 1 and use OR
- To flip a bit in a target bit pattern, set the corresponding mask bit to 1 and use the XOR operator