

บทที่ 4

การกระทำกับบิต (Operation on Bits)

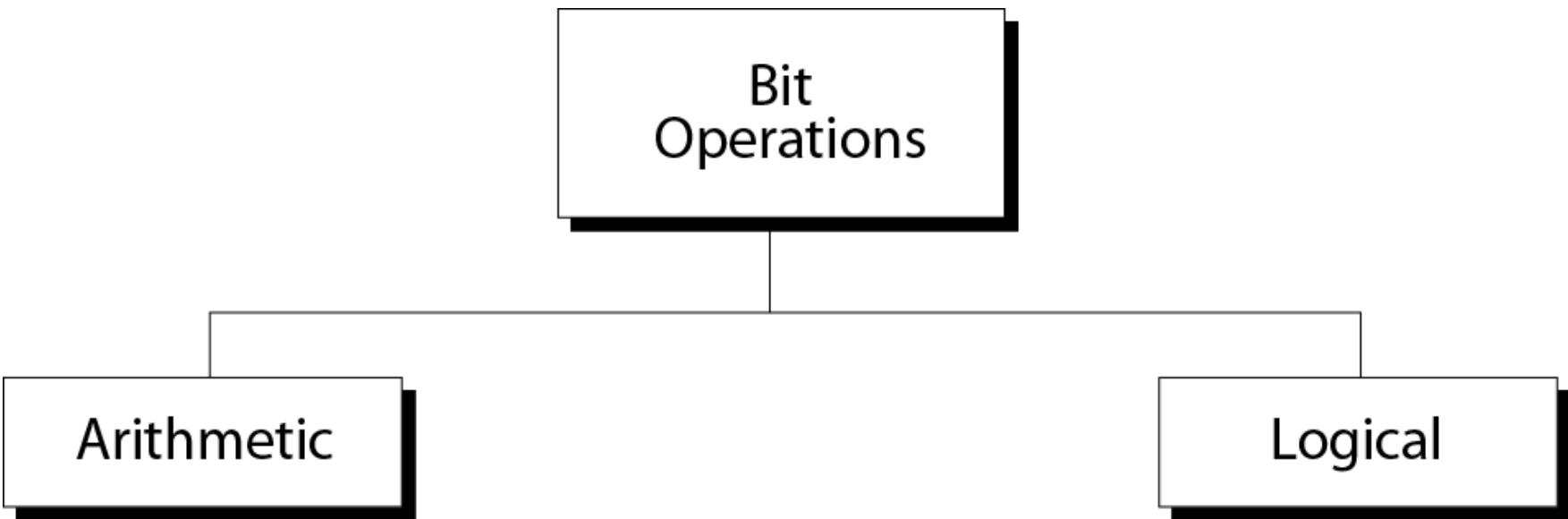
วัตถุประสงค์

หลังจากเรียนจบบทที่ 4 แล้ว นักศึกษาต้องสามารถ:

- ประยุกต์ arithmetic operations กับบิตได้เมื่อเลขจำนวนเต็มแทนด้วยรูปแบบ two's complement
- ประยุกต์ logical operations กับบิตได้
- เข้าใจการประยุกต์ logical operations โดยใช้ masks
- เข้าใจ shift operations กับเลขฐานสอง และเข้าใจการคูณ การหารเลขโดยการใช้ shift operations ของกำลังของเลขทั้งสองจำนวน

รูปที่ 4-1

การกระทำกับบิต



4.1

การกระทำเลขคณิต

การกระทำเลขคณิต

- **Arithmetic Operations** ประกอบด้วย การบวก การลบ การคูณ การหาร และอื่นๆอีก เราสามารถใช้การกระทำเหล่านี้กับ **จำนวนเต็ม** และ **จำนวนจริง** ได้
- สำหรับการกระทำกับจำนวนเต็มจะอธิบายเฉพาะการบวกและการลบเท่านั้น เพราะว่าการคูณสามารถทำได้โดยการใช้ซอฟท์แวร์ทำการบวกหลายครั้งหรืออาจใช้อาร์ดแวร์ก็ได้ ส่วนการหารก็สามารถใช้ซอฟท์แวร์ทำการลบหลายครั้งหรืออาจใช้อาร์ดแวร์ก็ได้เช่นกัน
- เราสามารถทำการบวกหรือลบกับจำนวนเต็มในทุกรูปแบบของการแทนแต่ในที่นี่จะอธิบายเฉพาะการแทนแบบ two's complement เท่านั้น

การบวกเลขจำนวนเต็มที่อยู่ในรูป two's complement

- การบวกเลขแบบ two's complement ก็เหมือนกับการบวกเลขฐาน 10 คือบวกทีละ column ถ้ามีตัวทด (carry) ก็จะนำไปบวกกับตัวลัดไป
- สิ่งที่จะต้องจดจำคือเรามาลังทำกับเลขฐาน 2 ไม่ใช่เลขฐาน 10 เมื่อเราบวกเลข 2 บิต ผลลัพธ์คือ 0 หรือ 1 เท่านั้น เมื่อมีการทด ตัวทดจะเท่ากับ 1 และนำไปบวกกับบิตที่อยู่ด้านซ้าย
- กฎที่ต้องระลึกเสมอสำหรับการบวกเลขจำนวนเต็มที่อยู่ในรูป two's complement คือ “**Add 2 bits and propagate the carry to the next column. If there is a final carry after the leftmost column addition, discard it.**”

ตารางที่ 4.1 การบวกกันของบิต

จำนวน 1	ผลลัพธ์	ตัวกด
None	0	
One	1	
Two	0	
Three	1	1

ตัวอย่างที่ 1

จงแสดงวิธีการบวกเลข two's complement ต่อไปนี้: (+17) + (+22) = (+39)

วิธีทำ

ตัวทด

1

$$\begin{array}{r} 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1 \\ + \\ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 1 \ 0 \end{array}$$

ผลลัพธ์

$$0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 1 \ 1 = 39$$

Example 2

จงแสดงการบวกเลขสองจำนวนต่อไปนี้ในรูปแบบที่แทนด้วย two's complement : (+24) + (-17) = (+7)

วิธีทำ

ตัวทด

1 1 1 1 1

0 0 0 1 1 0 0 0 +
1 1 1 0 1 1 1 1

ผลลัพธ์

0 0 0 0 0 1 1 1 = +7

ตัวอย่างที่ 3

จงแสดงการบวกเลขต่อไปนี้ในรูปแบบ two's complement:

$$(-35) + (+20) = (-15)$$

วิธีทำ

ตัวทด

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

ผลลัพธ์

$$1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 1 = -15$$

ตัวอย่างที่ 4

จงแสดงการบวกเลขต่อไปนี้ในรูปแบบ two's complement:

$$(+127) + (+3) = (+130)$$

วิธีทำ

ตัวทด

$$\begin{array}{r} 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \\ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ + \\ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \\ \hline \end{array}$$

ผลลัพธ์

$$1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 = -126 \text{ (ผิดพลาด)}$$

เกิด overflow

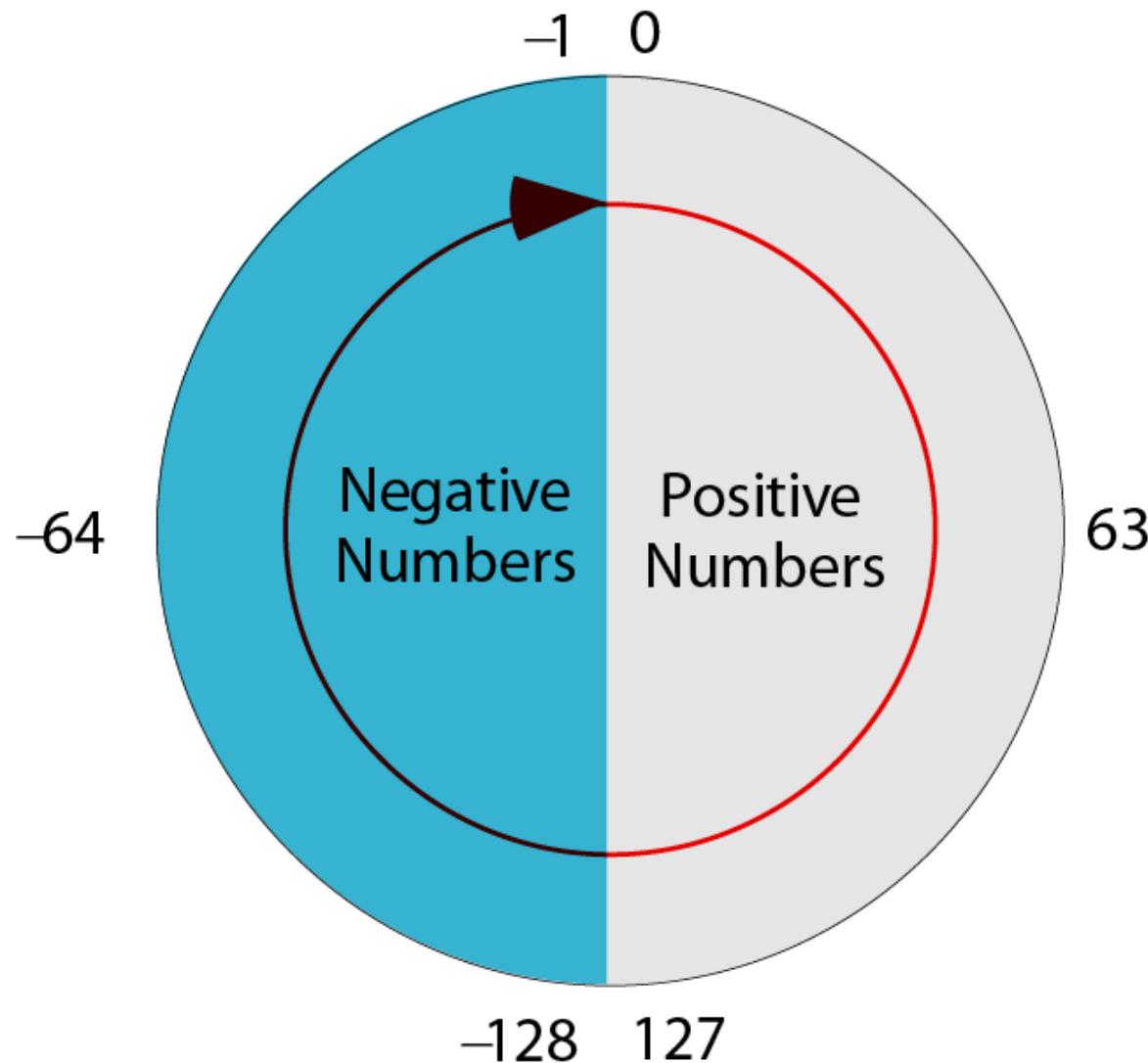
Overflow

- ถ้าให้ N แทนจำนวนบิตที่ใช้เก็บเลขจำนวนเต็ม 1 ค่าแล้วช่วงค่าของตัวเลขจำนวนเต็มที่อยู่ในรูป two's complement จะอยู่ระหว่าง

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

ตัวอย่างเช่นถ้า $N = 8$ ค่าจะอยู่ในช่วง -128 ถึง 127 จากตัวอย่างที่ 4 อธิบายว่าทำไม -126 จึงเกิด overflow ได้ดังนี้

ງົບໆ 4-2 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.

การลบเลขจำนวนเต็มที่อยู่ในรูป two's complement

- การลบเลขแบบ two's complement จะไม่แตกต่างจากการบวก นั้นคือ ทำการ negate (two's complement) ตัวที่จะลบมาแล้วจึงทำการบวก กันตามปกติธรรมดาก็ได้
- ทั้งนี้เพราะว่า

Number1 – Number2 มีค่าเท่ากับ Number1 + (-Number2)

เช่น

$$(+101) - (+62) = (+101) + (-62) = (+39)$$

ตัวอย่างที่ 5

จงแสดงการลบ 62 ออกจาก 101 โดยใช้รูปแบบการแทนตัวเลขเป็น two's complement: $(+101) - (+62)$ มีความหมายเหมือนกับ $(+101) + (-62)$

วิธีทำ

ตัวทด

1 1

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

$$0 0 1 0 0 1 1 1 = 39$$

ตัวทดซ้ายมือสุดตัดทิ้ง ไม่ต้องสนใจ

การกระทำเลขคณิต (+,-,x,/) กับเลขจำนวนจริง

- เราจะพิจารณาเฉพาะการบวกและการลบเท่านั้น (ด้วยเหตุผล เช่นเดียวกับจำนวนเต็ม) มีขั้นตอนดังนี้

ขั้นที่ 1: ตรวจสอบเครื่องหมาย

- ถ้าเครื่องหมายของเลขทั้งสองตัวเหมือนกัน ให้ทำ (การบวกเลขทั้งสองตัวนั้นแล้วได้เครื่องหมายเดิมที่ผลลัพธ์)
- ถ้าเครื่องหมายต่างกัน ให้เปรียบเทียบค่าสมบูรณ์ของเลขทั้งสองจากนั้นให้ลบเลขที่น้อยกว่าออกจากเลขที่มากกว่า แล้วได้เครื่องหมายของจำนวนที่มากกว่าไว้ที่ผลลัพธ์

การกระทำเลขคณิต (+,-,x,/) กับเลขจำนวนจริง (ต่อ)

ขั้นที่ 2: เลื่อนจุดศูนย์เพื่อทำให้ค่าเลขยกกำลัง (exponent) เท่ากัน
นั่นคือถ้าค่าเลขยกกำลังไม่เท่ากัน จุดศูนย์ของเลขที่มีค่าเลข
ยกกำลังน้อยกว่าจะถูกเลื่อน (shift) ไปทางซ้ายเพื่อทำให้ค่าเลข
ยกกำลังเท่ากัน

ขั้นที่ 3: บวกหรือลบค่า mantissas (ทั้ง whole part และ fraction part)

ขั้นที่ 4: ทำการ normalize ผลลัพธ์ก่อนนำไปเก็บในหน่วยความจำ

ขั้นที่ 5: ตรวจสอบ overflow

ตัวอย่างที่ 6

จงแสดงวิธีการบวกเลข floating-point 2 จำนวนต่อไปนี้:

0 10000100 10110000000000000000000000000000

0 10000010 01100000000000000000000000000000

วิธีทำ

ขั้นที่ 1 ค่าเลขยกกำลังของเลขที่กำหนดคือ 5 และ 3 ดังนั้น เลขทั้งสองเขียน

ในรูป normalization คือ $+2^5 \times 1.1011$ และ $+2^3 \times 1.011$

ขั้นที่ 2 นำค่าเลขยกกำลังของเลขทั้งสองจำนวนให้เท่ากัน จะได้

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

ขั้นที่ 3 หลังจากทำการ normalization จะได้ $= +2^6 \times 1.000001$

ขั้นที่ 4 จัดเก็บในหน่วยความจำเป็น:

0 10000101 00000100000000000000000000000000

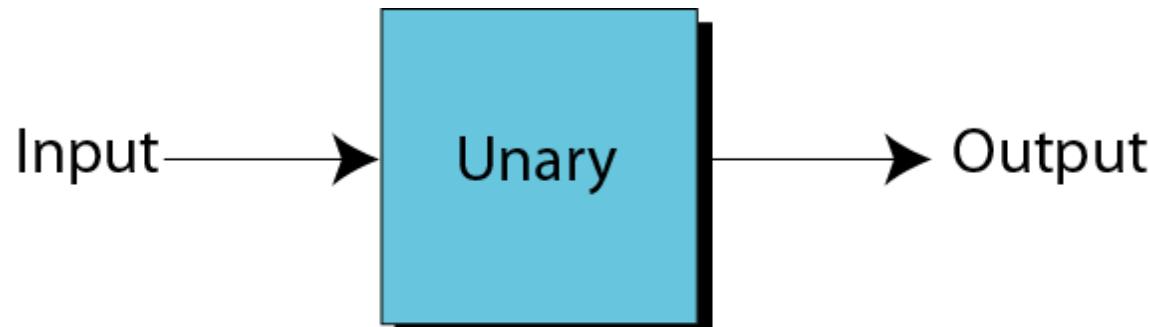
4.2

การกระทำตรรกะ
LOGICAL OPERATIONS

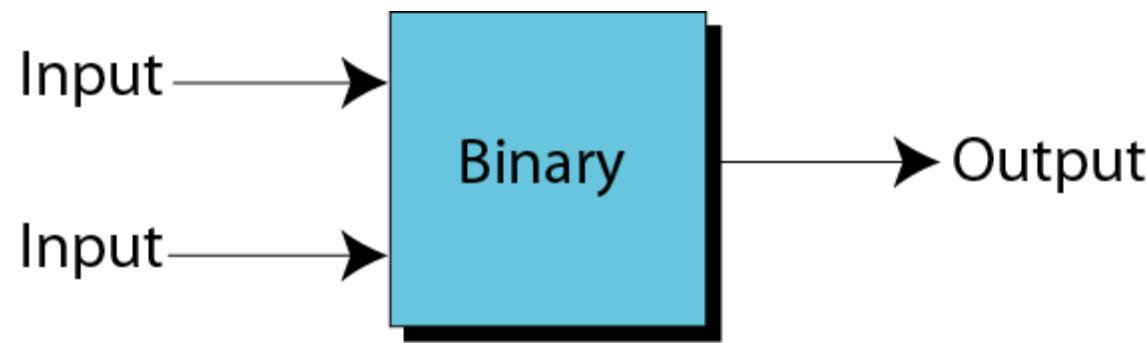
การกระทำตรรกะ

- ค่าของบิตอาจเป็น 0 หรือ 1 เราอาจตีความหมายว่า **0** คือ **เท็จ** (false) ส่วน **1** คือ **จริง** (true) ดังนั้น 1 บิตที่เก็บในหน่วยความจำอาจแทนค่า ตรรกะที่เป็นจริงหรือเท็จ
- Logical operation อาจใช้ 1 หรือ 2 บิตเป็นอินพุทเพื่อก่อให้เกิดค่า เอาท์พุท 1 บิต ถ้าการกระทำใช้ 1 บิต จะเรียกการกระทำนี้ว่า **unary operation** แต่ถ้าการกระทำใช้ 2 บิต จะเรียกการกระทำนี้ว่า **binary operation**

Unary and binary operations



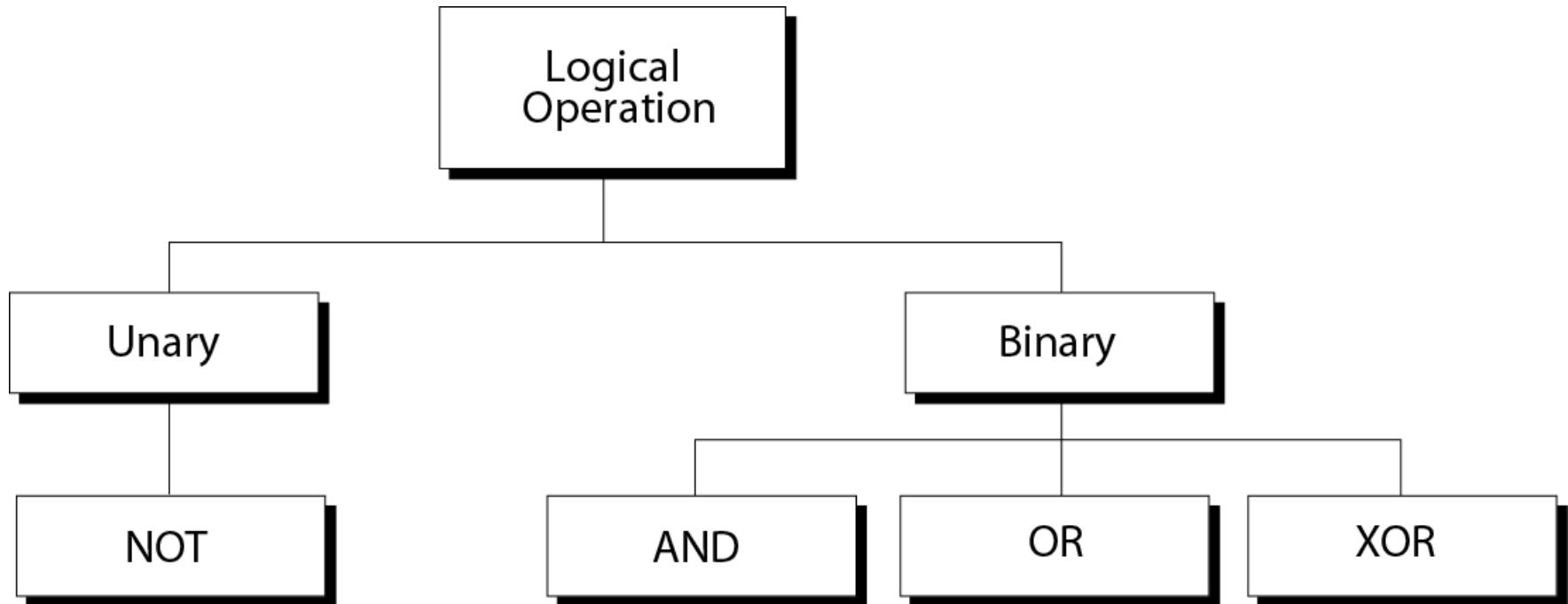
a. Unary operator



b. Binary operator

รูปที่ 4-4

การกระทำตรรก



Truth tables

NOT

x	NOTx
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

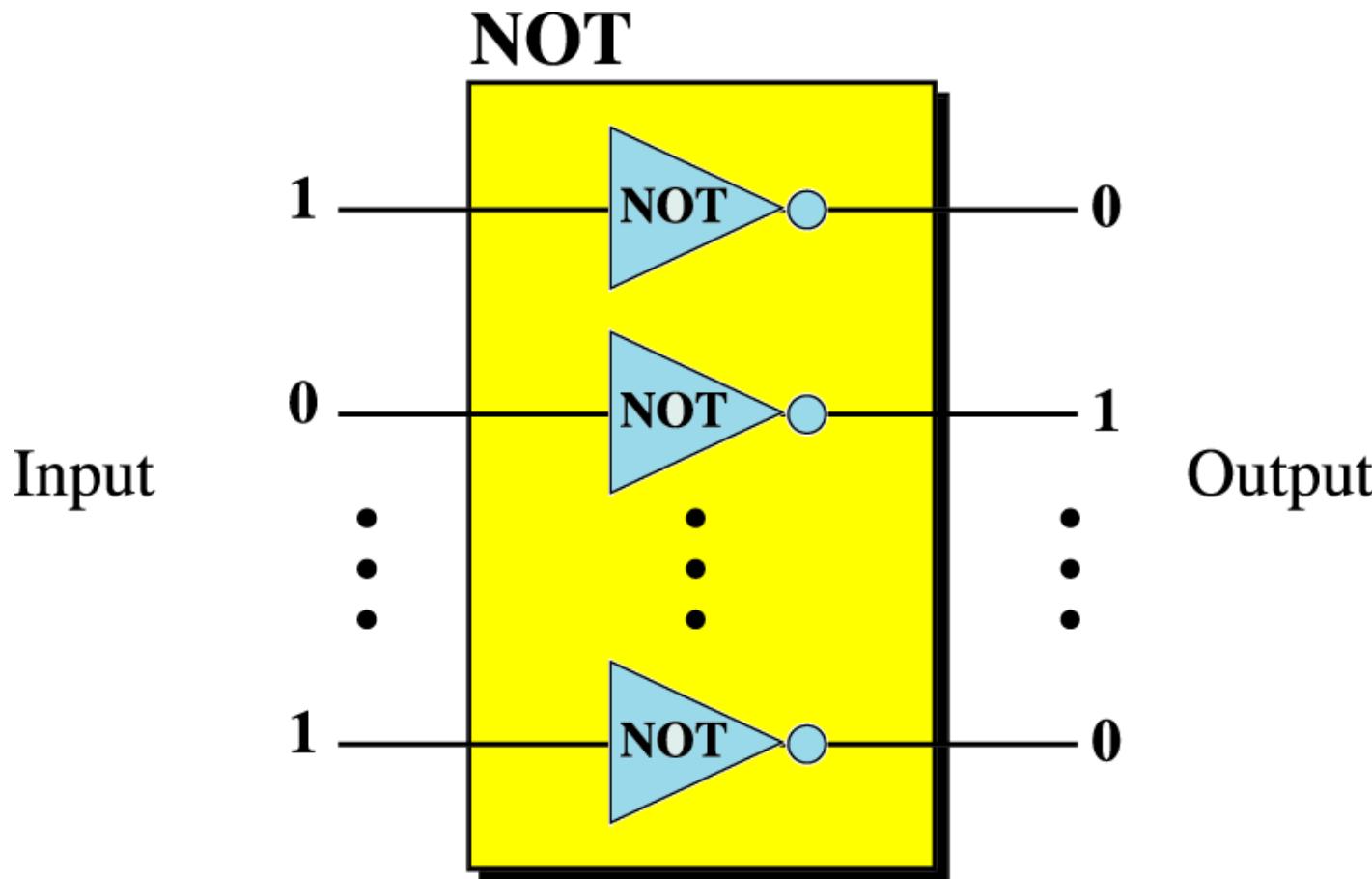
XOR

x	y	x XORy
0	0	0
0	1	1
1	0	1
1	1	0



ตารางที่ 4-6

การกระทำ NOT



ตัวอย่างที่ 7

จงใช้การกระทำ NOT กับ bit pattern 10011000

วิธีทำ

ตัวตั้ง

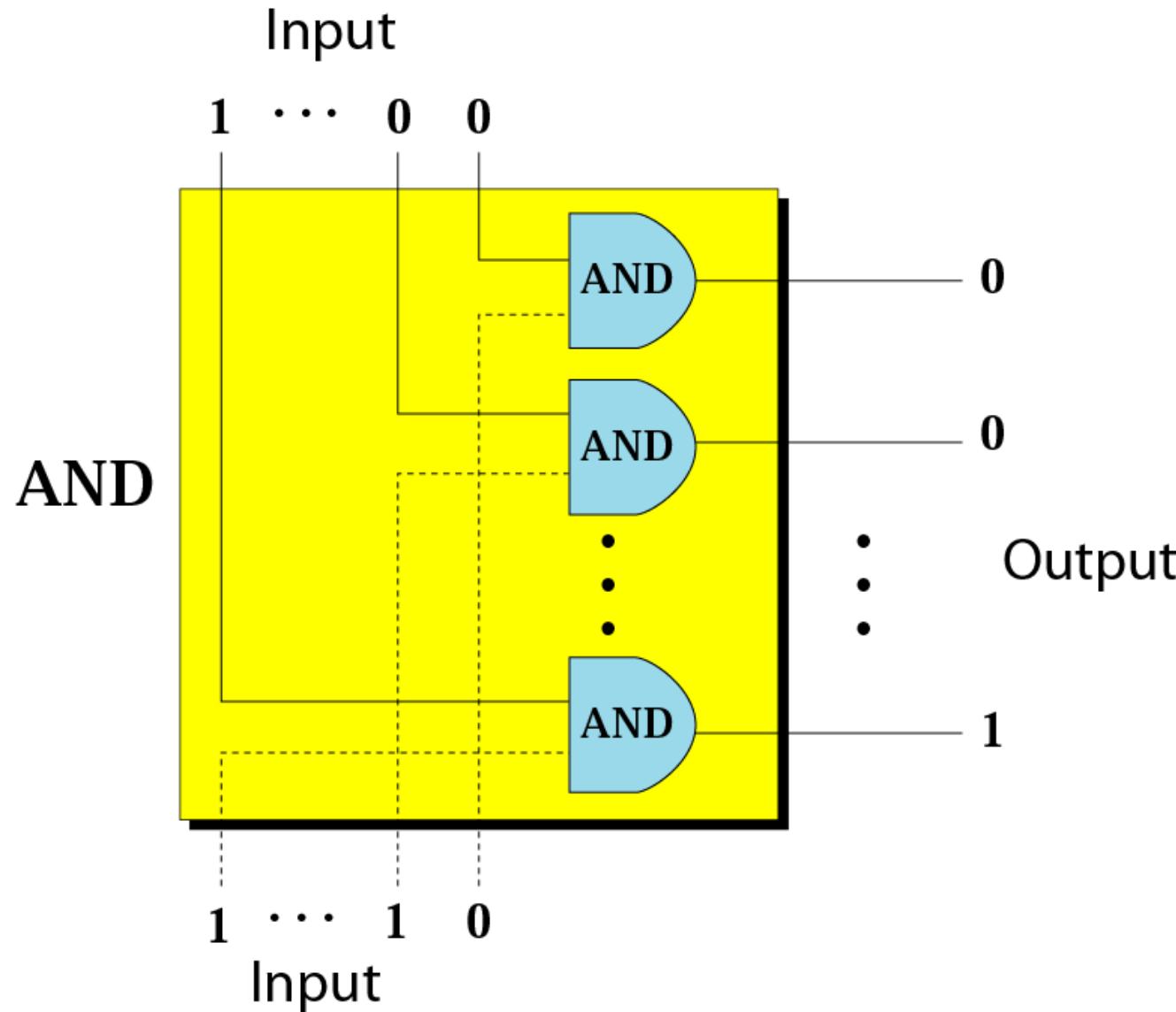
1 0 0 1 1 0 0 0 NOT

0 1 1 0 0 1 1 1

ผลลัพธ์

รูปที่ 4-7

การกระทำ AND



ตัวอย่างที่ 8

จงใช้การกระทำ AND ระหว่าง bit patterns 10011000 กับ bit patterns 00110101

วิธีทำ

ตัวตั้ง

1 0 0 1 1 0 0 0 AND

0 0 1 1 0 1 0 1

0 0 0 1 0 0 0 0

ผลลัพธ์

รูปที่ 4-8

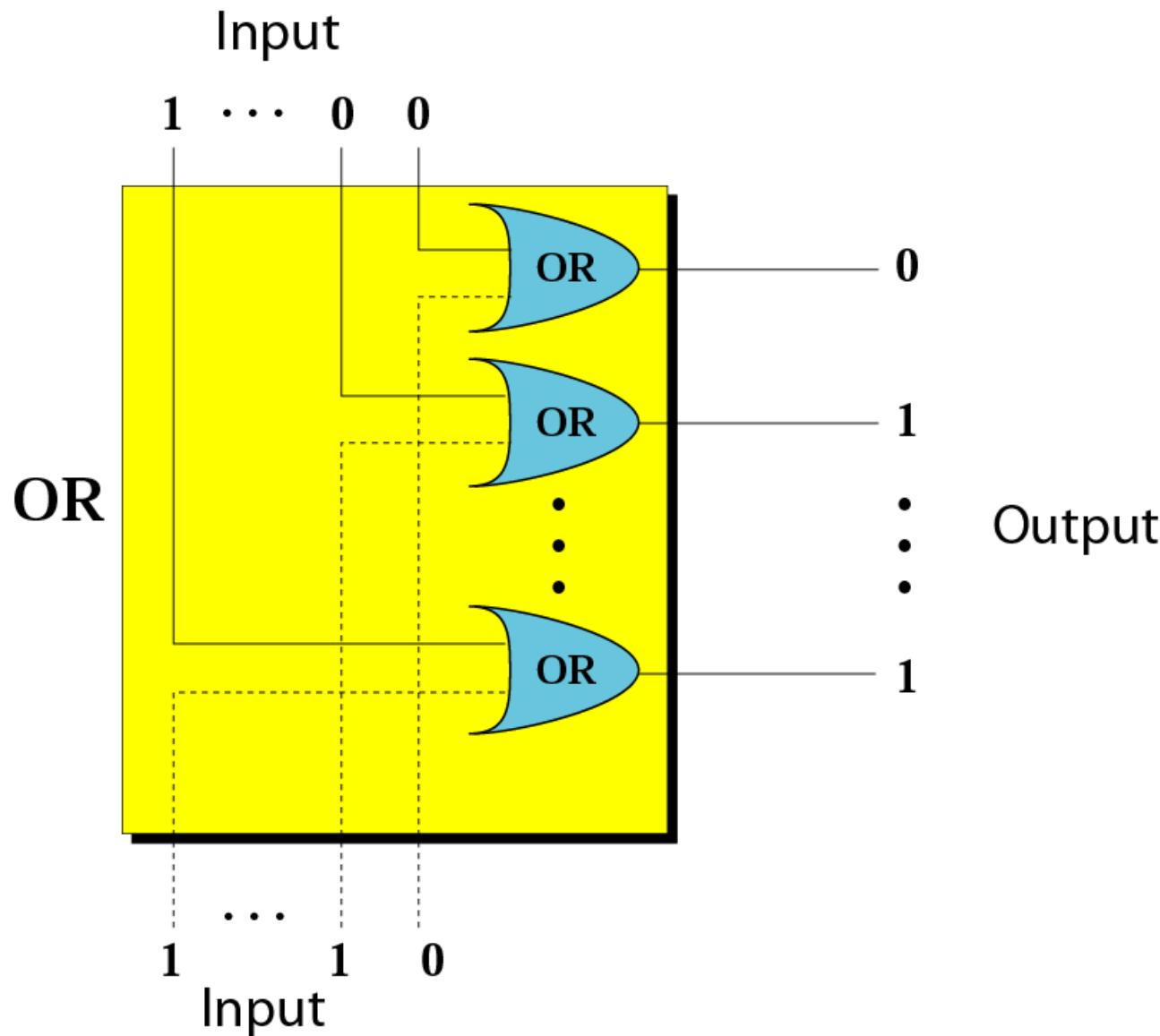
กฎของการกระทำ AND

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

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

รูปที่ 4-9

การกระทำ OR



ตัวอย่างที่ 9

จงใช้การกระทำ OR กระทำระหว่าง bit pattern 10011000 กับ bit pattern 00110101

วิธีทำ

ตัวตั้ง

1 0 0 1 1 0 0 0 OR

0 0 1 1 0 1 0 1

1 0 1 1 1 1 0 1

ผลลัพธ์

รูปที่ 4-10

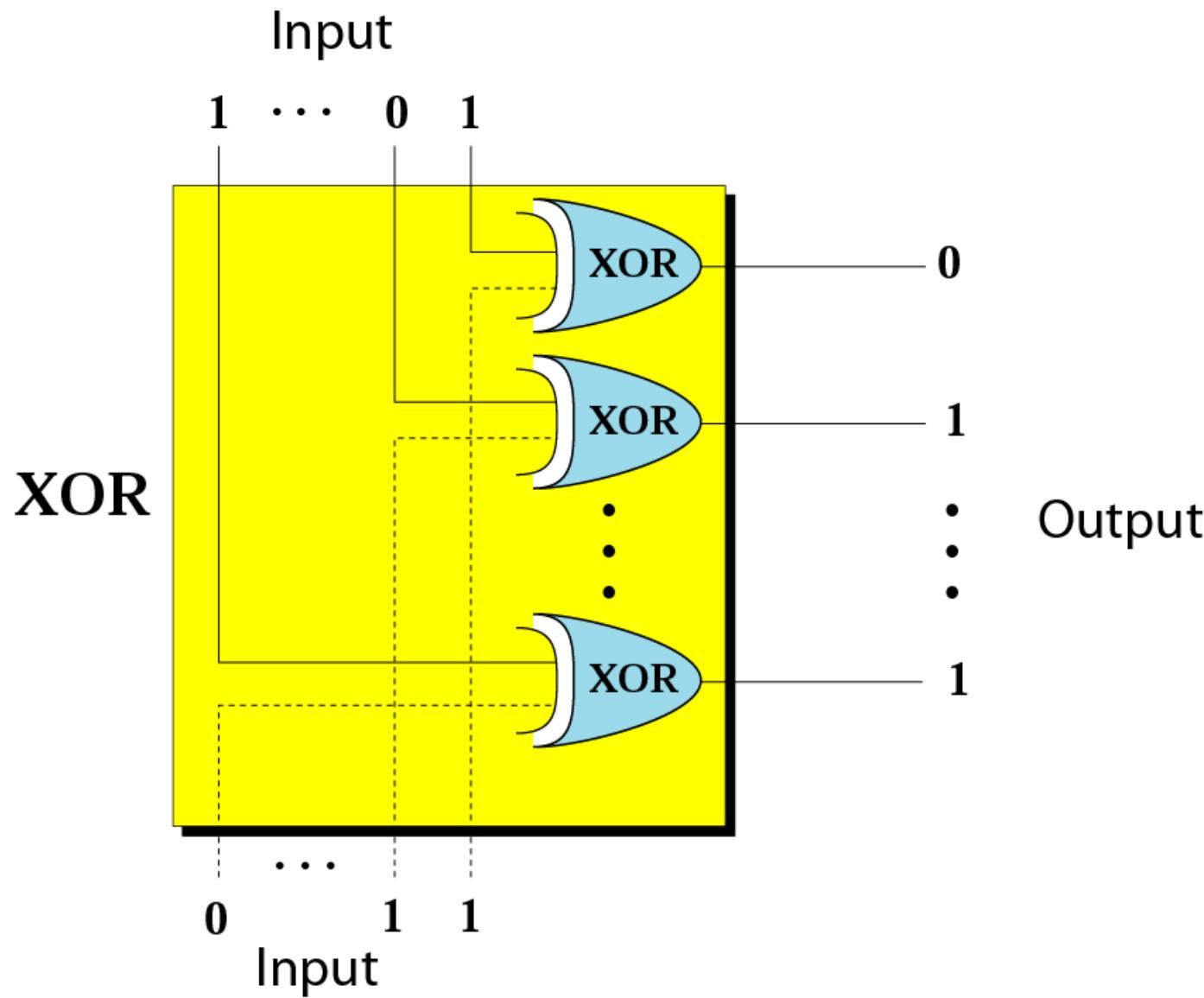
กฎของการรวม OR

$$(1) \text{ OR } (X) \longrightarrow (1)$$

$$(X) \text{ OR } (1) \longrightarrow (1)$$

รูปที่ 4-11

การกระทำ XOR



ตัวอย่างที่ 10

จงใช้การกระทำ XOR กระทำกับ bit pattern 10011000 และ bit pattern 00110101

วิธีทำ

ตัวตั้ง

1 0 0 1 1 0 0 0 XOR

0 0 1 1 0 1 0 1

1 0 1 0 1 1 0 1

ผลลัพธ์



Brooks/Cole
Thomson Learning™

©Brooks/Cole, 2003

รูปที่ 4-12

กฎของการกระทำ XOR

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

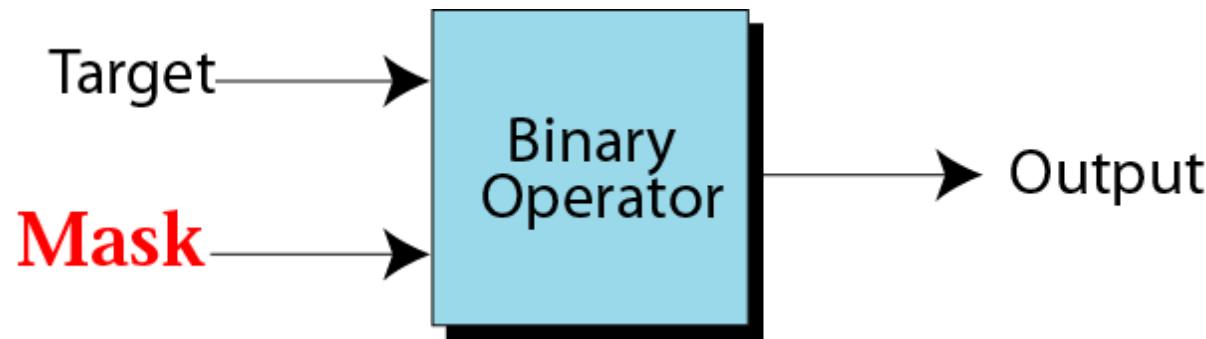
$$(X) \quad \text{XOR} \quad (1) \quad \longrightarrow \quad \text{NOT}(X)$$

การประยุกต์

- การกระทำตระหง่านสามประเภทสามารถนำไปใช้ปรับเปลี่ยน **bit pattern** ได้โดยการ unset, set, หรือ reverse specific bits.
- Bit pattern ที่จะทำการปรับเปลี่ยนคือทำการ ANDed, ORed, หรือ XORed กับอีก bit pattern หนึ่งซึ่งเรียกว่า **mask**
- Bit pattern ที่เรียกว่า mask นี้แหลบที่เราใช้สำหรับปรับเปลี่ยน bit pattern อันอื่น

រូបភាព 4-13

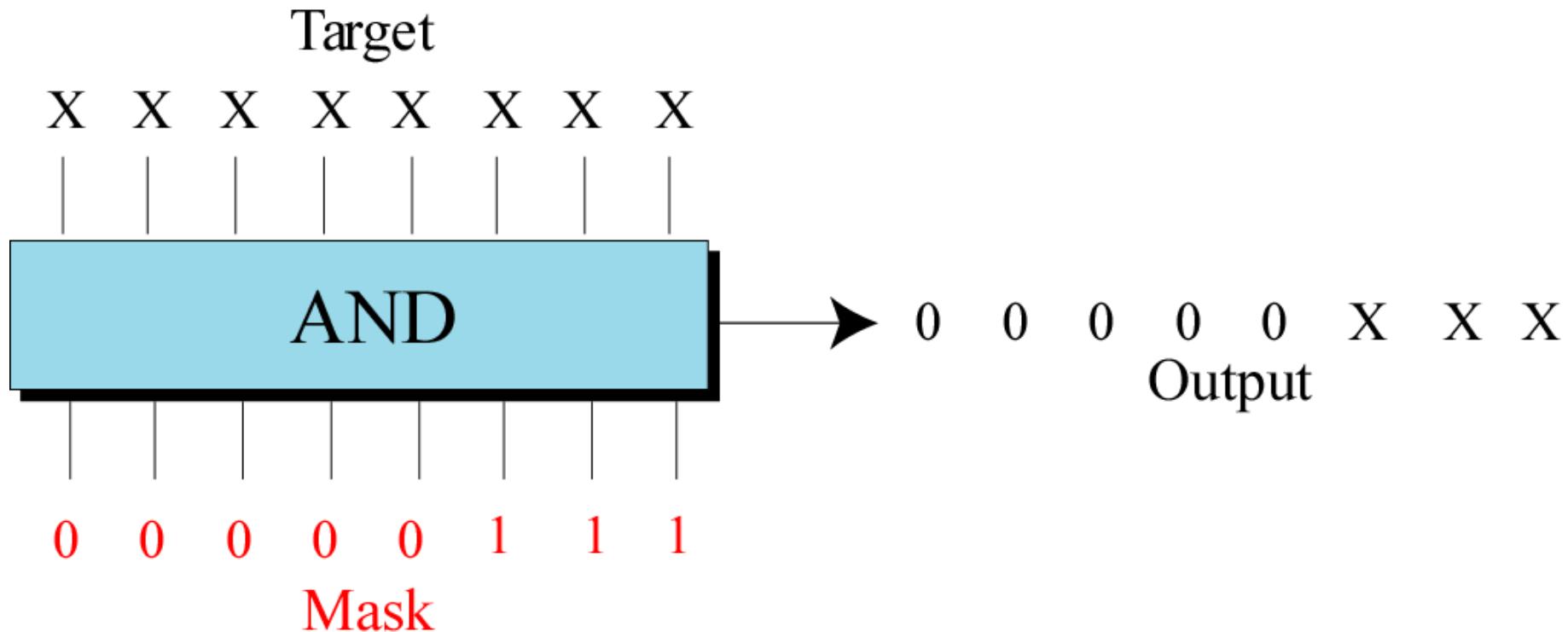
Mask



Unsetting Specific Bits

- การประยุกต์อย่างหนึ่งของ operation **AND** คือใช้ทำ unset (set ให้บิตที่ต้องการใน bit pattern เป็น 0) ซึ่งทำได้โดยการใช้ unsetting mask ที่มีจำนวนบิตเท่ากัน กฎในการสร้าง unsetting mask สรุปได้ดังนี้
 - ถ้าต้องการให้บิตเป้าหมายเป็น 0 ให้กำหนดบิต 0 ใน mask ณ ตำแหน่งที่ตรงกับบิตเป้าหมาย
 - ถ้าต้องการให้บิตเป้าหมายคงเดิม ให้กำหนดบิต 1 ใน mask ณ ตำแหน่งที่ตรงกับบิตเป้าหมาย

รูปที่ 4-14 ตัวอย่าง unsetting specific bits



ตัวอย่างที่ 11

จงหา mask เพื่อ unset (ทำให้เป็น 0) 5 บิตทางซ้ายของ pattern 10100110

ວິທີທຳ

Mask គឺ ០ ០ ០ ០ ០ ១ ១ ១

๕๖

1 0 1 0 0 1 1 0

AND

Mask

0 0 0 0 0 1 1 1

ผลลัพธ์

0 0 0 0 0 1 1 0



ตัวอย่างที่ 12

สมมติว่ากรุงเทพมหานครมีเครื่องปั๊มน้ำจำนวน 8 เครื่องเพื่อสูบน้ำออกสู่ทະเล สถานะของปั๊ม (on หรือ off) สามารถแทนได้ด้วย bit pattern ขนาด 8 บิต เช่น pattern 11000111 และงว่าปั๊ม 1, 2, 3 (จากทางขวา) 7, และ 8 เปิดอยู่ ในขณะที่ปั๊ม 4, 5, และ 6 นั้นปิดอยู่ สมมติต่อไปว่าเราต้องการปิดปั๊มที่ 7 จงแสดงให้เห็นว่า mask จะช่วยแก้ปัญหานี้ได้อย่างไร?

วิธีทำอยู่หน้าตัดไป

วิธีทำ

กำหนดให้ mask **10111111** เพื่อทำการ AND กับ pattern เป้าหมาย
บิตที่ 7 ใน mask ที่เป็น 0 จะทำให้บิตที่ 7 ใน pattern เป้าหมายเป็น
0 และได้ดังนี้

เป้าหมาย

1 1 0 0 0 1 1 1 AND

Mask

1 0 1 1 1 1 1 1

ผลลัพธ์

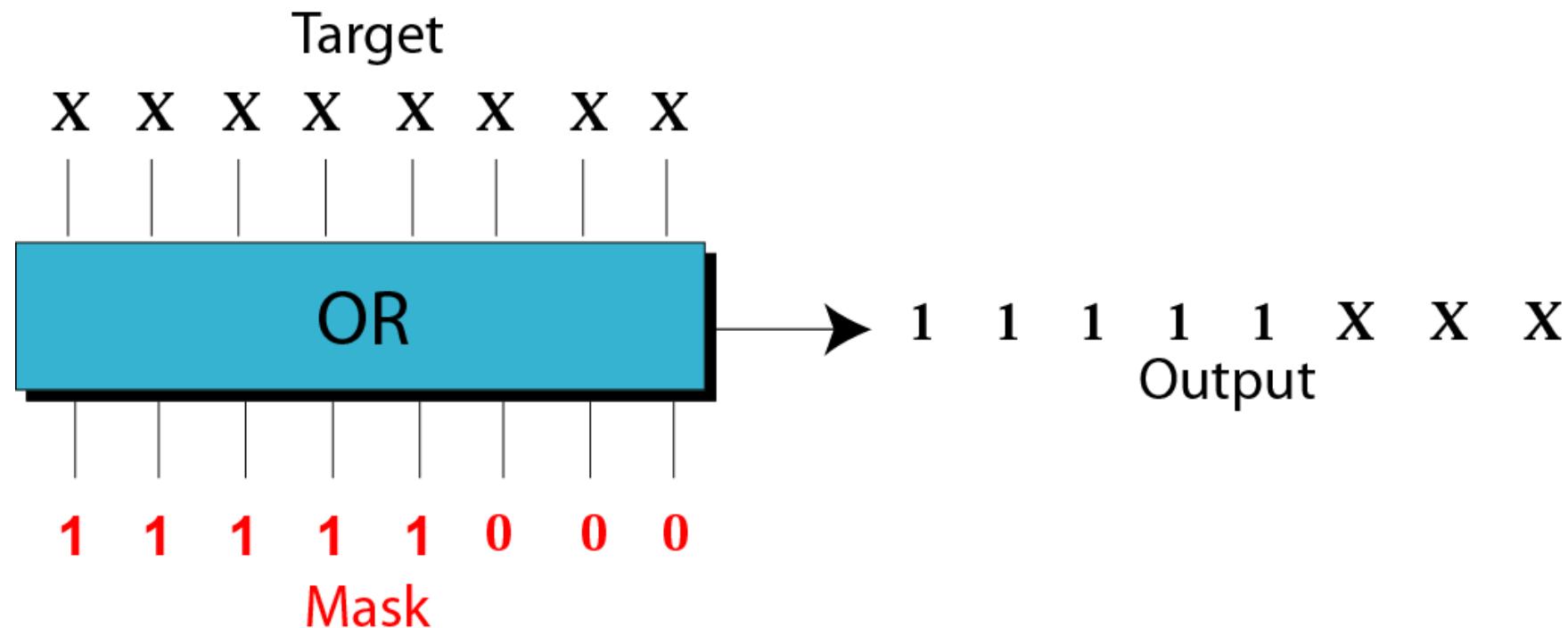
1 0 0 0 0 1 1 1

Setting Specific Bits

- บกประยุกต์อย่างหนึ่งของการกระทำ **OR** คือการ **set** (กำหนดให้บิตเป้าหมายเป็น 1) ซึ่งทำได้โดยการใช้ **setting mask** ที่มีจำนวนบิตเท่ากัน
- กฎการสร้าง **setting mask** มีดังนี้
 - ถ้าต้องการให้บิตเป้าหมายเป็น 1 ให้กำหนดบิต 1 ใน mask ณ ตำแหน่งที่ตรงกับบิตเป้าหมาย
 - ถ้าต้องการให้บิตเป้าหมายคงเดิม ให้กำหนดบิต 0 ใน mask ณ ตำแหน่งที่ตรงกับบิตเป้าหมาย

รูปที่ 4-15

ตัวอย่างของ setting specific bits



ตัวอย่างที่ 13

จงหา mask เพื่อทำการ set 5 บิตทางซ้ายมือของ pattern 10100110 ให้เป็น 1

วิธีทำ

กำหนดให้ Mask คือ **1 1 1 1 1 0 0 0**

pattern

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

ตัวอย่างที่ 14

จากปั๊มน้ำในตัวอย่างที่ 12 จงหาว่าจะมีวิธีเปิดปั๊มที่ 6 ได้อย่างไร?

วิธีทำ

ใช้ mask **00100000.**

เข้าหมาย

1 0 0 0 0 1 1 1

OR

0 0 1 0 0 0 0 0

1 0 1 0 0 1 1 1

ผลลัพธ์



Brooks/Cole
Thomson Learning™

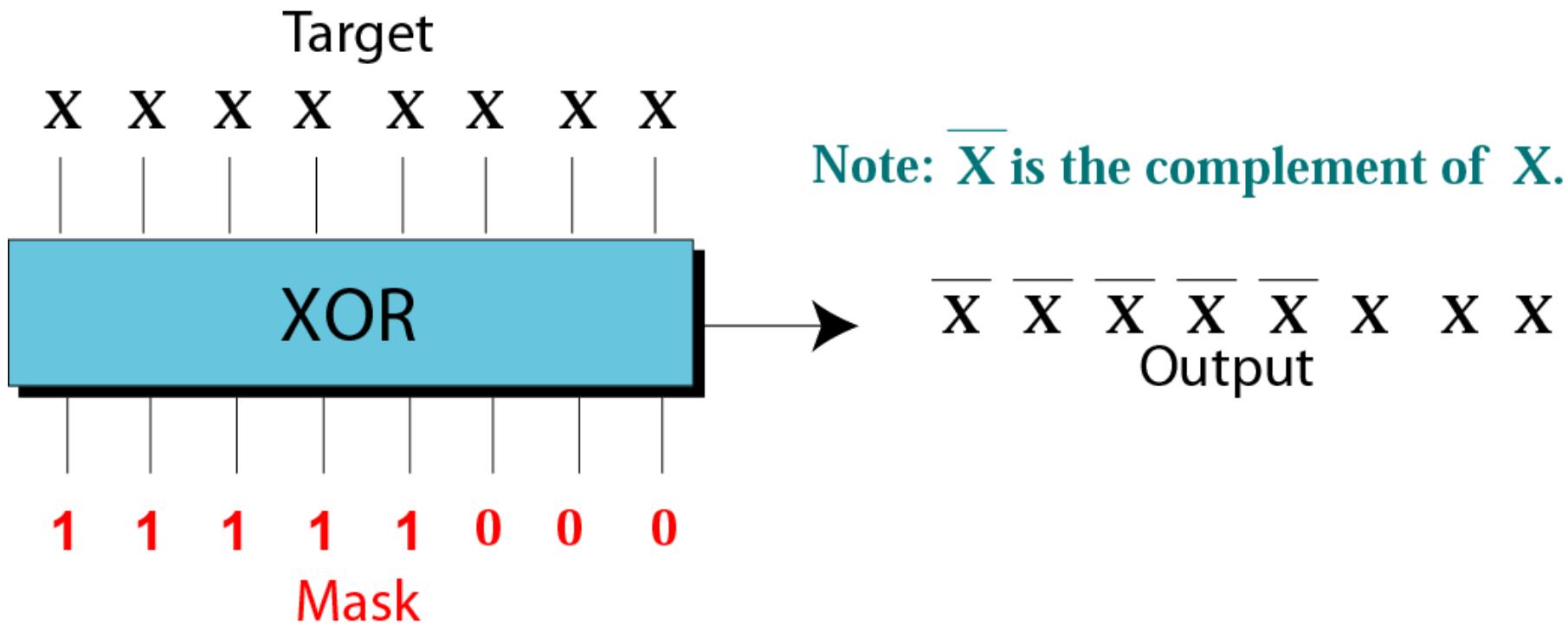
©Brooks/Cole, 2003

Flipping Specific Bits

- บทประยุกต์อันหนึ่งของ XOR คือการเปลี่ยนค่าของบิตให้เป็นตรงกันข้าม (flip) หมายความว่าเปลี่ยนค่าบิต ณ ตำแหน่งที่ต้องการจาก 0 เป็น 1 หรือจาก 1 เป็น 0
- กฎในการสร้าง XOR mask มีดังนี้
 - ถ้าจะเปลี่ยนค่าบิต ณ ตำแหน่งที่ต้องการเป็นตรงกันข้าม ให้กำหนดค่าบิตเท่ากับ 1 ณ ตำแหน่งที่ตรงกันใน mask
 - ถ้าจะปล่อยให้ค่าบิต ณ ตำแหน่งที่ต้องการไม่เปลี่ยนแปลง ให้กำหนดค่าบิตเท่ากับ 0 ณ ตำแหน่งที่ตรงกันใน mask

รูปที่ 4-16

ตัวอย่างการ flip specific bits



ตัวอย่างที่ 15

จงสร้าง mask เพื่อเปลี่ยน 5 บิตแรกของ pattern เป็นตรงกันข้าม
ทดสอบ mask กับ pattern 10100110.

วิธีทำ

เป้าหมาย

1 0 1 0 0 1 1 0

XOR

Mask

1 1 1 1 1 0 0 0

0 1 0 1 1 1 1 0

ผลลัพธ์



4.3

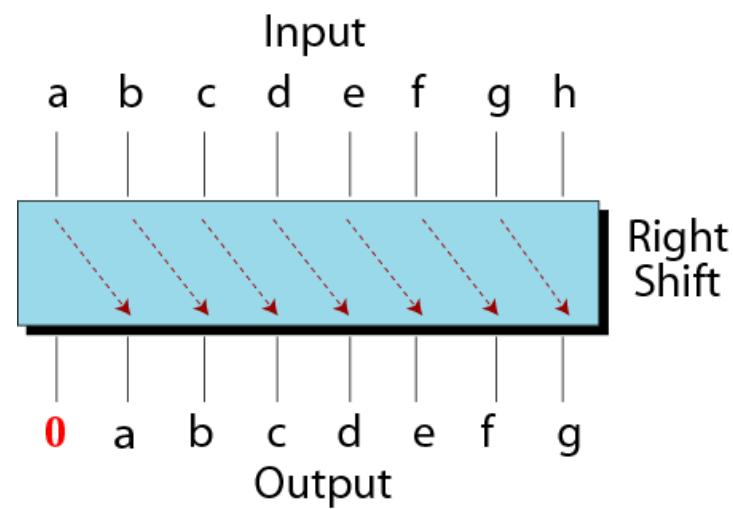
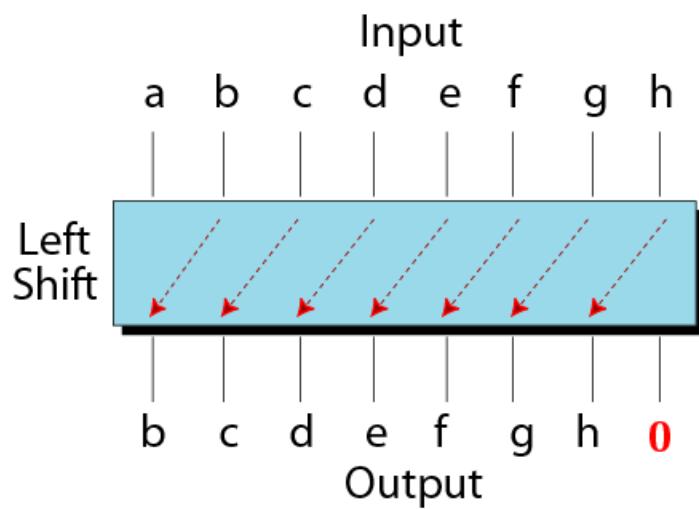
SHIFT OPERATIONS

Shift Operations

- Operation ที่สำคัญอีกอันหนึ่งสำหรับ bit pattern คือ shift
- การ shift สามารถทำได้ 2 ลักษณะคือ shift ไปทางซ้าย กับ shift ไปทางขวา
- การ shift ไปทางขวาจะทำให้บิตขวาสุดหายไปแล้วจึงเลื่อนทุกๆบิตไปทางขวา 1 ตำแหน่ง และทำการเพิ่มบิต 0 เข้าไปที่ตำแหน่งซ้ายมือสุด
- การ shift ไปทางซ้ายจะทำให้บิตซ้ายสุดหายไปแล้วจึงเลื่อนทุกๆบิตไปทางซ้าย 1 ตำแหน่ง และทำการเพิ่มบิต 0 เข้าไปที่ตำแหน่งขวาสุด
- Shift operations จะใช้กับ unsigned numbers เท่านั้น

รูปที่ 4-17

Shift operations



ตัวอย่างที่ 16

จงแสดงให้เห็นว่าท่านสามารถคูณหรือหารเลขด้วย 2 โดยใช้ shift

operations.

วิธีทำ

ถ้า bit pattern แทนเลข unsigned การ shift ขวา 1 ครั้ง เท่ากับการหารด้วย 2 เช่น pattern 00111011 แทนจำนวนเต็ม 59 เมื่อ shift ไปทางขวา 1 ครั้งจะได้ 00011101 ซึ่งเท่ากับ 29 แต่ถ้า shift bit pattern เดิมไปทางซ้าย 1 ครั้งจะได้ 01110110 ซึ่งเท่ากับ 118 นั้นคือเหมือนกับคูณด้วย 2 นั้นเอง

ตัวอย่างที่ 17

จะใช้การกระทำตระกะและการกระทำ shift เพื่อหาค่า (0 or 1) ของบิตที่ 4 จากทางขวาของ bit pattern

วิธีทำ

ใช้ mask 00001000 ไป AND กับ pattern เป้าหมายเพื่อจะคงสถานะของบิตที่ 4 ไว้แต่ทำให้บิตอื่นๆเป็น 0

รายละเอียดอยู่ในสไลด์ต่อไป

วิธีทำ (ต่อ)

เข้าหมาย

abcdefgh AND

Mask

00001000

ผลลัพธ์

0000e000

ทำการ shift ผลลัพธ์ 3 ครั้งไปทางขวาดังนี้

0000e000 → 00000e00 → 000000e0 → 0000000e

ณ จุดนี้จะพบว่าง่ายต่อการตรวจสอบค่าของ pattern ใหม่ที่ได้จากการ shift ซึ่งถือว่าเป็นจำนวนเต็มแบบ unsigned ถ้าค่าตัวเลขเป็น 1 แสดงว่าค่าบิตดังเดิมเท่ากับ 1 ถ้าค่าตัวเลขเป็น 0 และแสดงว่าบิตเดิมเป็น 0

คำสำคัญในบทที่ 4

- | | | | |
|---------------------------|--------------------------|---------------------|--------------|
| • AND operator | OR operator | NOT operator | |
| • Binary operation | Binary operator | Carry | Clear |
| • Flip | Logical operation | Mask | Set |
| • Unary operation | Unary operator | unset | |