

## บทที่ 3

# การแทนจำนวนเลข ในเครื่องคอมพิวเตอร์

# วัตถุประสงค์

---

หลังจากเรียนจบบทที่ 3 แล้ว นักศึกษาต้องสามารถ:

- เปลี่ยนเลขฐานไปมาระหว่างเลขฐาน 2 กับเลขฐาน 10 ได้
- เข้าใจความแตกต่างของการแทนเลขจำนวนเต็มภายในเครื่องคอมพิวเตอร์แบบ unsigned, sign-and-magnitude, one's complement, และ two's complement.
- เข้าใจระบบ Excess ที่ใช้เก็บค่ายกกำลังของเลขจำนวนจริง
- เข้าใจว่าจำนวนจริงเก็บในเครื่องคอมพิวเตอร์อย่างไรโดยใช้ exponent และ mantissa

# บทนำ

- รูปแบบการเก็บข้อมูลที่เป็นตัวเลขในคอมพิวเตอร์มีความแตกต่างจากรูปแบบการเก็บข้อมูลที่ไม่ใช่ตัวเลข (nonnumeric data) เหตุผลที่ต้องจัดเก็บด้วยรูปแบบที่ต่างกันมีดังนี้
  - รหัสที่ใช้แทนตัวอักษร เช่น ASCII ไม่เหมาะสมที่จะใช้แทนตัวเลข ASCII สามารถใช้แทนสัญลักษณ์ได้ 128 ตัว แต่สัญลักษณ์ที่เป็นตัวเลข มีเพียง 10 ตัวเท่านั้น เช่นถ้าต้องการแทนจำนวน 65,535 โดยใช้ รหัส ASCII เราต้องการเพียง 5 ไบท์เท่านั้น (1 ไบท์ต่อ 1 ตัว) แต่ถ้าตัวเลข ดังกล่าวแทนโดยใช้ unsigned integer และ จะใช้เพียง 2 ไบท์เท่านั้น

# บทนำ

2. **การกระทำ** (operation) ที่เกี่ยวกับตัวเลข เช่น บวก ลบ คูณ หาร จะยุ่งยากมากถ้าตัวเลขแต่ละตัวแทนด้วยรหัสที่เป็นตัวอักษร
3. **การแทนระดับความถูกต้อง**ของตัวเลข (precision) ที่แสดงหลังจุดทศนิยมนั้นใช้หน่วยความจำในการจัดเก็บมาก (byte intensive) เช่น การเก็บตัวเลข 23454.00001 จะต้องใช้ถึง 11 ไบท์ แต่ถ้าแทนด้วยรูปแบบที่เรียกว่า floating point แล้วจะใช้จำนวนไบท์ที่น้อยกว่ามาก
  - \* ด้วยเหตุผลที่กล่าวมาเป็นเพียงส่วนหนึ่ง จึงจะต้องแยกรูปแบบการเก็บตัวเลขมาไว้ต่างหากโดยเฉพาะ

**3.1**

## เลขฐาน 10 และเลขฐาน 2

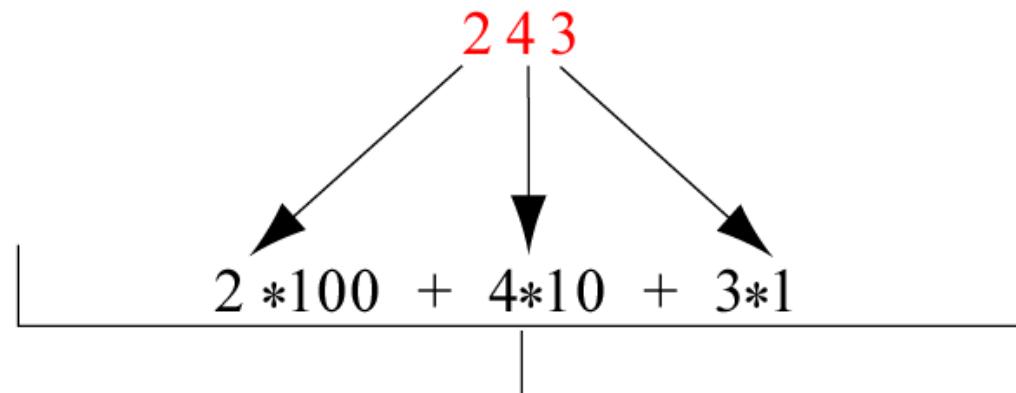
# เลขฐานสิบและเลขฐานสอง

- ระบบตัวเลข 2 ระบบที่ใช้กันมากในอุตสาหกรรมคอมพิวเตอร์คือ **ระบบเลขฐาน 10 (decimal)** และ **ระบบเลขฐาน 2 (binary)**
- ระบบเลขฐาน 10: กิดค้นโดยนักคณิตศาสตร์ชาวอาเรเบียน (Arabian) ในศตวรรษที่ 8 ปัจจุบันใช้กันทั่วโลกในทุกวิธีการ
- มนุษย์ชนชาติแรกที่ใช้ระบบเลขฐาน 10 คือชาวอียิปต์โบราณ หลังจากนั้นชาวบabilon (Babylonians) ได้ปรับปรุงระบบตัวเลขของชาวอียิปต์ให้มีความหมายมากขึ้นโดยการกำหนดให้ตัวหน่วยของตัวเลขมีความหมาย นั่นคือตัวหน่วยจากทางขวา มีจะเป็นหลักหน่วย หลักสิบ หลักร้อย ...

# Decimal system

|        |        |        |        |        |
|--------|--------|--------|--------|--------|
| $10^4$ | $10^3$ | $10^2$ | $10^1$ | $10^0$ |
| 10,000 | 1000   | 100    | 10     | 1      |

Decimal Positions



Two Hundred Forty-Three

# เลขฐานสิบและเลขฐานสอง

- ระบบเลขฐาน 2: คล้ายกับระบบ decimal แต่ฐานเป็น 2 โดยมีเลขเพียง 2 ตัวคือ 0 กับ 1 ตำแหน่งของตัวเลขแต่ละตัวต่างกันมีความหมายเช่นกัน ดังในรูปหน้าต่อไป

# Binary system

|       |       |       |       |       |       |       |       |
|-------|-------|-------|-------|-------|-------|-------|-------|
| $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |
| 128   | 64    | 32    | 16    | 8     | 4     | 2     | 1     |

Binary Positions

1 1 1 1 0 0 1 1

$$1*128 + 1*64 + 1*32 + 1*16 + 0*8 + 0*4 + 1*2 + 1*1$$

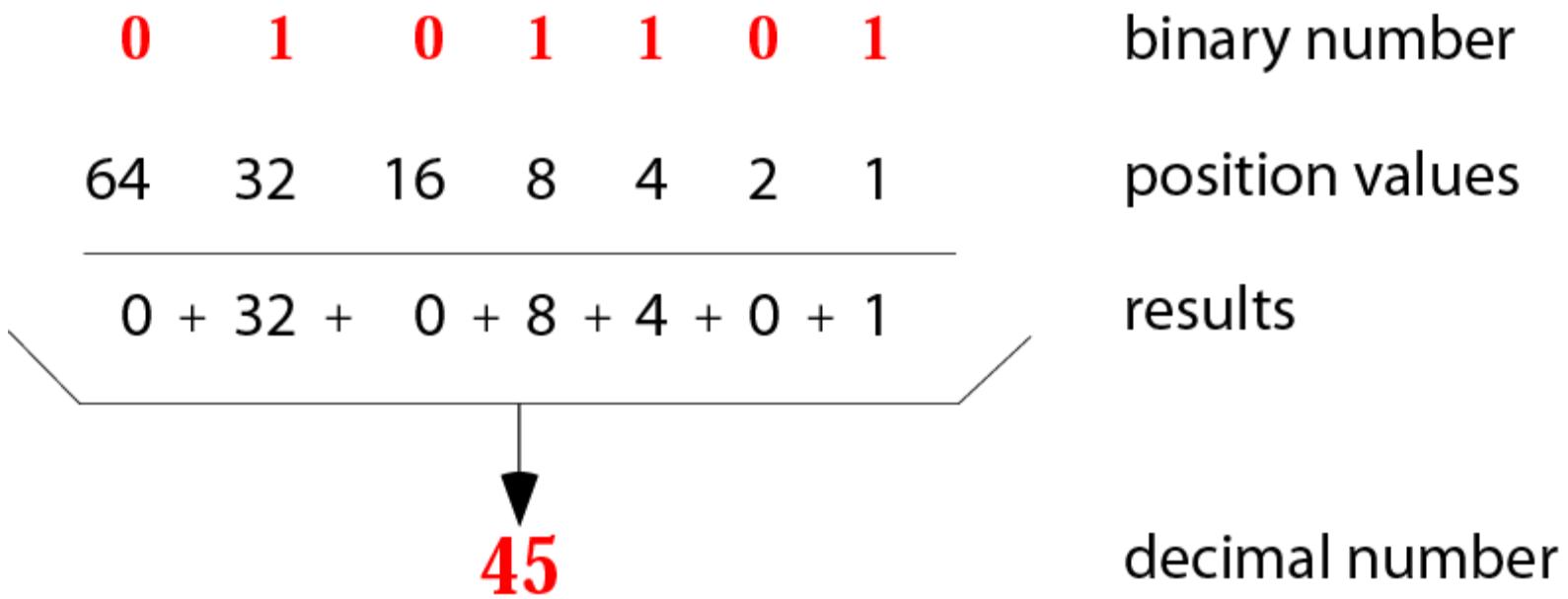
Two Hundred Forty-Three

**3.2**

## การแปลงเลขฐาน

### รูปที่ 3-3

## การแปลงจากเลขฐาน 2 เป็นเลขฐาน 10



# ตัวอย่างที่ 1

จะเปลี่ยนเลขฐานสอง 10011 เป็นเลขฐานสิบ

วิธีทำ

ขั้นแรกเขียนนำหน้าหักภายในตัวบิตของเลขฐานสอง ขั้นที่สองทำการคูณค่าของบิตกับนำหน้าและบันทึกผลลัพธ์ไว้ ขั้นสุดท้ายบวกผลลัพธ์ทั้งหมดจะได้เลขฐานสิบที่ต้องการ

|           |    |   |   |   |   |
|-----------|----|---|---|---|---|
| เลขฐานสอง | 1  | 0 | 0 | 1 | 1 |
| นำหน้า    | 16 | 8 | 4 | 2 | 1 |

$$16 + 0 + 0 + 2 + 1$$

เลขฐานสิบ

19

## ตัวอย่างที่ 2

จงเปลี่ยน 35 (ฐานสิบ) เป็นเลขฐานสอง

### วิธีทำ

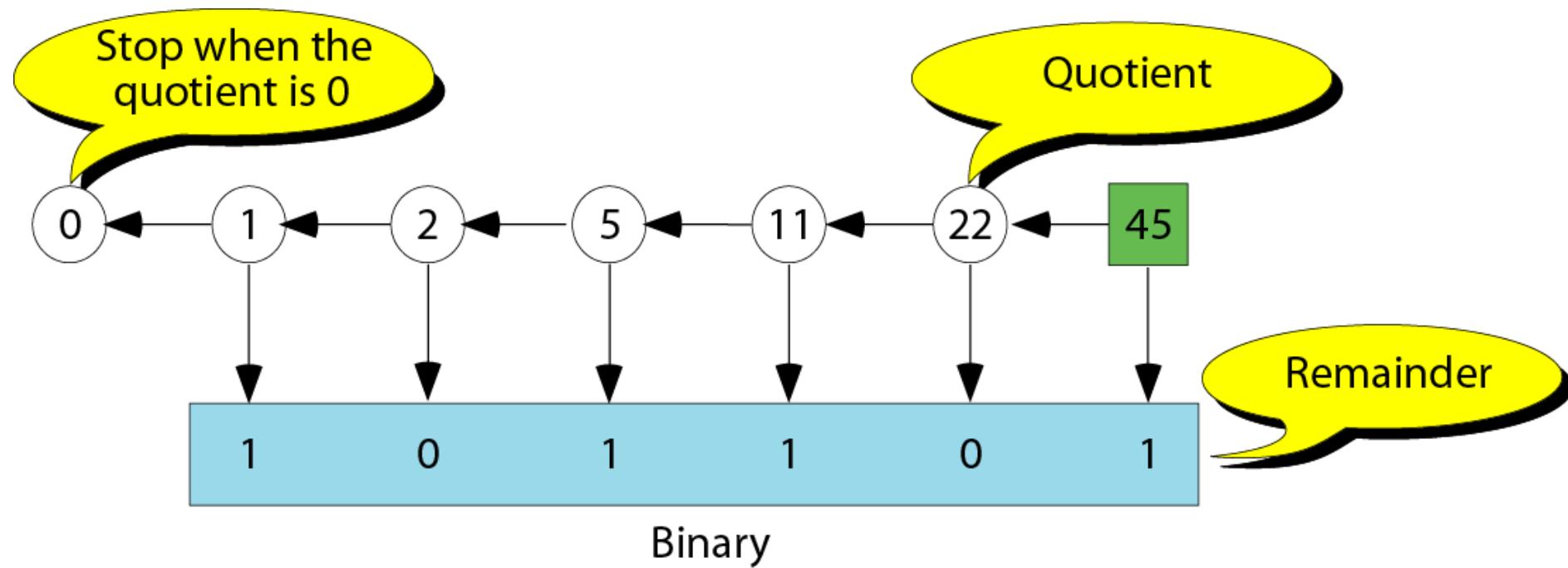
วิธีการเปลี่ยนมีหลายวิธี แต่จะอธิบายวิธีที่เข้าใจง่ายที่สุดคือเริ่มต้นด้วยการเขียนเลขฐาน 10 ที่ต้องการเปลี่ยนไว้ทางขวา จากนั้นเอา 2 หารเลขจำนวนนั้นไปเรื่อยๆ การหารแต่ละครั้งให้เขียนผลหารและเศษจากการหารโดยเขียนผลหารทางด้านซ้าย และเขียนเศษใต้ผลหาร การหารจะหยุดเมื่อผลการหารเท่ากับศูนย์ ดังตัวอย่างข้างล่าง

0 ← 1 ← 2 ← 4 ← 8 ← 17 ← 35 เลขฐานสิบ

เลขฐานสอง

1 0 0 0 1 1

## การเปลี่ยนเลขฐานสิบเป็นเลขฐานสอง



**3.4**

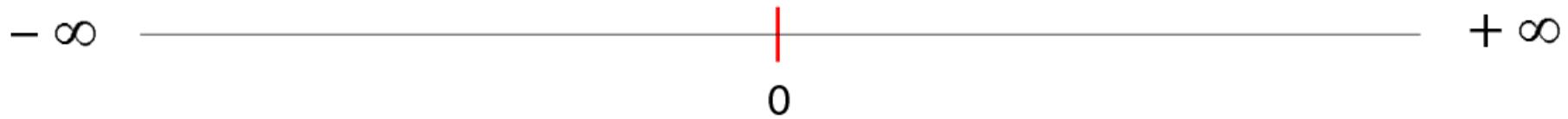
## การแทนเลขจำนวนเต็ม

### 3.4.1 พิสัยของจำนวนเต็ม

จากหัวข้อที่แล้วเราทราบวิธีการแปลงจากเลขฐาน 10 เป็นเลขฐาน 2 ไปแล้ว ในหัวข้อนี้จะศึกษารูปแบบการเก็บจำนวนเต็มในหน่วยความจำของคอมพิวเตอร์ จำนวนเต็มอาจเป็น **เลขบวก** หรือเป็น **เลขลบ** จำนวนเต็มลบมีค่าตั้งแต่  $-\infty$  ถึง 0 ส่วนจำนวนเต็มบวกมีค่าตั้งแต่ 0 ถึง  $+\infty$  (รูปที่ 3.5) อย่างไรก็ดี ไม่มีคอมพิวเตอร์เครื่องใดในโลกที่สามารถเก็บเลขที่มีขนาดใหญ่เช่นนี้ได้ เพราะจำนวนบิตที่ใช้จะต้องเป็น **อนันต์ (infinity)** นั่นคือเครื่องคอมพิวเตอร์จะต้องมีขนาดของหน่วยความจำเป็นอนันต์นั่นเอง การเก็บจำนวนเต็มในหน่วยความจำแบ่งรูปแบบการเก็บดังรูปที่ 3.6

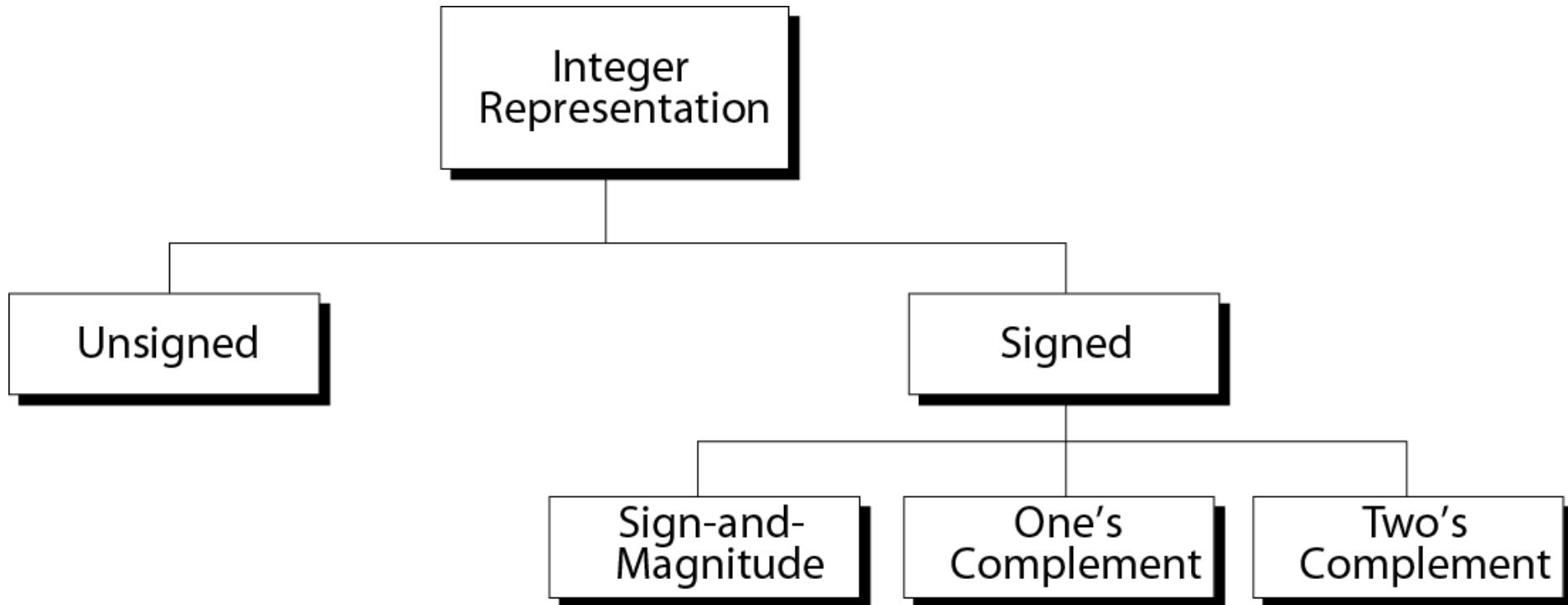
# รูปที่ 3-5

## พิสัยของเลขจำนวนเต็ม



## รูปที่ 3-6

# ประเภทของการจัดเก็บจำนวนเต็ม



# รูปแบบการเก็บ Unsigned Integers

- **Unsigned integer** คือจำนวนเต็มที่ไม่มีเครื่องหมาย มีค่าอยู่ระหว่าง 0 ถึง อนันต์ (infinity)
- แต่ไม่มีคอมพิวเตอร์เครื่องใดที่จะสามารถเก็บค่าของจำนวนเต็ม ในช่วงดังกล่าวได้ ปกติเครื่องคอมพิวเตอร์แต่ละเครื่องจะกำหนดค่าคงที่ที่มีค่ามากที่สุด (maxint) ดังนั้นค่า unsigned integers จึงมีค่าอยู่ระหว่าง 0 ถึง ค่ามากสุดที่กำหนด (maxint)
- ค่ามากสุดที่กำหนดขึ้นอยู่กับจำนวนบิตที่คอมพิวเตอร์ใช้เก็บ
- ถ้า  $N = \text{จำนวนบิต}$  ค่าจะอยู่ระหว่าง 0 ...  $(2^N - 1)$

## ตารางที่ 3.1 พิสัยของ unsigned integers

| จำนวนบิต | พิสัย          |
|----------|----------------|
| -----    | -----          |
| 8        | 0 ..... 255    |
| 16       | 0 ..... 65,535 |

# ขั้นตอนการแทน **unsigned integers**

- การจัดเก็บ **unsigned integers** มีขั้นตอนดังนี้
  - ขั้นที่ 1: เปลี่ยนตัวเลขที่ต้องการเก็บให้เป็นเลขฐาน 2
  - ขั้นที่ 2: ถ้าจำนวนบิตของเลขฐาน 2 น้อยกว่า N ให้เติม 0 เข้าไปที่ตำแหน่งบิตทางซ้ายของเลขฐาน 2 ที่ได้จากข้อ 1 จนครบทั้งหมด N บิต

## ตัวอย่างที่ 3

จงจัดเก็บ 7 ในหน่วยความจำ 8 บิต

### วิธีทำ

ขั้นที่ 1 แปลง 7 ให้เป็นเลขฐานสองก่อนได้ 111

ขั้นที่ 2 เนื่องจากมีแค่ 3 บิตจึงต้องเติม 0 เข้าไปทางซ้ายอีก 5 ตัว

เพื่อให้เต็ม 8 ตัว จะได้ **00000111**

ขั้นที่ 3 เก็บ **00000111** ในหน่วยความจำของคอมพิวเตอร์

## ตัวอย่างที่ 4

จงเก็บเลข 258 ในหน่วยความจำ 16 บิต

### วิธีทำ

ขั้นที่ 1 แปลง 258 ให้เป็นเลขฐานสองก่อนได้ 100000010

ขั้นที่ 2 เนื่องจากมีแค่ 9 บิตจึงต้องเติม 0 เข้าไปทางซ้ายอีก 7 ตัว

เพื่อให้เต็ม 16 ตัว จะได้ **0000000100000010**

ขั้นที่ 3 เก็บ **0000000100000010** ในหน่วยความจำของคอมพิวเตอร์

## ตารางที่ 3.2 ตัวอย่างการจัดเก็บ unsigned integers ในหน่วยความจำขนาด 8 บิต และ 16 บิต

| เลขฐาน 10 | หน่วยความจำ 8 บิต | หน่วยความจำ 16 บิต |
|-----------|-------------------|--------------------|
| 7         | 00000111          | 000000000000111    |
| 234       | 11101010          | 000000011101010    |
| 258       | overflow          | 000000100000010    |
| 24,760    | overflow          | 011000010111000    |
| 1,245,678 | overflow          | overflow           |

## ตัวอย่างที่ 5

จงแปลงเลข unsigned integer 00101011 ให้เป็นเลขฐานสิบ

### วิธีทำ

ใช้วิธีการตามรูปที่ 3.3 จะได้เลขฐานสิบเท่ากับ 43.

# Overflow

- ถ้าเราพยายามจะเก็บ unsigned integer **ที่มีค่ามากกว่า 256** ด้วย **หน่วยความจำ 8 บิต** จะพบว่าค่า 256 มีค่ามากกว่าค่าของเลขที่หน่วยความจำจะสามารถจัดเก็บได้ จึงไม่สามารถจัดเก็บได้ **สภาพการเช่นนี้เรียกว่า overflow**
- หมายเหตุ: สำหรับหน่วยความจำที่ใช้ 8 บิตเก็บ unsigned integers ถ้าพยายามจะเก็บค่าที่มากกว่าหรือเท่ากับ 256 จะก่อให้เกิด overflow เสมอ

# การประยุกต์ใช้ unsigned integers

การเก็บ unsigned integers ในหน่วยความจำถือเป็นการใช้หน่วยความอย่างมีประสิทธิภาพ เพราะไม่ต้องเก็บเครื่องหมาย นั่นคือจำนวนบิตทั้งหมดที่ใช้จะใช้เพื่อการแทนค่าของตัวเลขเท่านั้น เราใช้ unsigned integers ในสถานการณ์ดังนี้

- \* **การนับ (counting)** เมื่อเราต้องการนับอะไรก็ตาม เราจะไม่ใช้เลขติดลบ ปกติเราจะเริ่มนับจาก 1 (บางครั้งก็เริ่มจาก 0 )
- \* **การกำหนดตำแหน่งที่อยู่ (addressing)** มีภาษาคอมพิวเตอร์บางภาษาที่เก็บที่อยู่ของตำแหน่งหน่วยความจำไว้เป็นข้อมูลในหน่วยความจำอื่น ตำแหน่งที่อยู่เป็นเลขจำนวนเต็มบวกเริ่มตั้งแต่ 0 (ไปที่แรกใน memory)

# การเก็บจำนวนเต็มแบบ sign-and-magnitude

- การเก็บจำนวนเต็มแบบ sign-and-magnitude ต้องใช้ 1 บิตเพื่อแทนเครื่องหมาย ( $0$  แทน  $+$ ,  $1$  แทน  $-$ ) หมายความว่าถ้าเราใช้หน่วยความจำจำนวน  $1$  ใบที่เก็บจำนวนเต็ม จริงๆแล้วเราใช้แค่  $7$  บิตเท่านั้นที่เก็บค่าของตัวเลขนั้น (ค่าที่ไม่มีเครื่องหมาย)
- ดังนั้นค่าจำนวนเต็มมากสูงสุดที่หน่วยความจำ  $1$  ใบจะเก็บได้โดยใช้รูปแบบ sign-and-magnitude จะมีค่าเท่ากับครึ่งหนึ่งของค่าจำนวนเต็มที่เป็น unsigned integer
- ถ้า  $N$  แทนจำนวนบิตที่ใช้แทนจำนวนเต็มแบบ sign-and-magnitude ค่าตัวเลขที่แทนได้จะอยู่ในช่วง  $-(2^{N-1} - 1)$  ถึง  $+(2^{N-1} - 1)$

# การเก็บเลข 0 แบบ sign-and-magnitude

- มีเลข 0 อよ' 2 แบบในการเก็บแบบ sign-and-magnitude คือ 0 ที่เป็นบวก และ 0 ที่เป็นลบ
- ถ้าเราใช้การจัดสารรแบบ 8 บิตเพื่อเก็บจำนวนเต็มแบบ sign-and-magnitude การเก็บ 0 ทั้งสองแบบจะมีลักษณะดังนี้

$+0 = 00000000$

$-0 = 10000000$

### ตารางที่ 3.3 พิลัยของ sign-and-magnitude integers

| จำนวนบิต | พิลัย          |       |       |                |
|----------|----------------|-------|-------|----------------|
| -----    | -----          | ----- | ----- | -----          |
| 8        | -127           | -0    | +0    | +127           |
| 16       | -32767         | -0    | +0    | +32767         |
| 32       | -2,147,483,647 | -0    | +0    | +2,147,483,647 |

# ขั้นตอนการเก็บจำนวนเต็มแบบ sign-and-magnitude

- การจัดเก็บจำนวนเต็มแบบ sign-and-magnitude มีขั้นตอนดังนี้
  - ขั้นที่ 1: แปลงเลขจำนวนเต็มให้เป็นแบบเลขฐาน 2 (ไม่คิดเครื่องหมาย)
  - ขั้นที่ 2: ถ้าจำนวนบิตของเลขฐาน 2 ที่ได้มีน้อยกว่า  $N-1$  บิต ให้ใส่ 0 ทางด้านซ้ายของตัวเลขจนครบ  $N-1$  บิต
  - ขั้นที่ 3: ถ้าจำนวนเป็นบวก ให้เพิ่ม 0 ที่บิตซ้ายสุด(ทำให้ครบ  $N$  บิต) แต่ถ้าจำนวนเป็นลบ ให้เพิ่ม 1 ที่บิตซ้ายสุด(ทำให้ครบ  $N$  บิต)



Note:

In sign-and-magnitude 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.

## ตัวอย่างที่ 6

จงหาค่า +7 ในหน่วยความจำ 8-บิตที่จัดเก็บแบบ sign-and-magnitude.

วิธีทำ

ขั้นที่ 1 เปลี่ยน +7 ให้เป็นเลขฐานสองได้ = 111

ขั้นที่ 2 เพิ่ม 0 จำนวน 4 ตัวเข้าไปข้างหน้าเลขฐานสองที่ได้เพื่อให้ครบ 7

บิตจะได้ = 0000111

ขั้นที่ 3 เพิ่ม 0 เข้าไปข้างหน้าอีก 1 ตัวเพื่อแสดงว่าเป็นเลขบวก ผลลัพธ์ที่ได้คือ: 00000111

## ตัวอย่างที่ 7

จงจัดเก็บ -258 ในหน่วยความจำขนาด 16-บิตโดยใช้การแทนแบบ sign-and-magnitude.

### วิธีทำ

ขั้นที่ 1 เปลี่ยน -258 ให้เป็นเลขฐานสองได้ = 100000010.

ขั้นที่ 2 เพิ่ม 0 เข้าไปข้างหน้า 6 ตัวเพื่อทำให้ครบ 15 ตัวได้ =

000000100000010.

ขั้นที่ 3 เพิ่ม 1 เข้าไปข้างหน้าเพื่อแสดงว่าเป็นเลขลบได้ =

**1000000100000010**

## ตารางที่ 3.4 ตัวอย่างการเก็บจำนวนเต็มแบบ sign-and-magnitude ด้วยขนาดหน่วยความจำที่ต่างกัน

| จำนวนเต็ม | หน่วยความจำขนาด 8 บิต | หน่วยความจำขนาด 16 บิต |
|-----------|-----------------------|------------------------|
| -----     | -----                 | -----                  |
| +7        | 00000111              | 0000000000000111       |
| -124      | 11111100              | 1000000011111100       |
| +258      | overflow              | 000000100000010        |
| -24,760   | overflow              | 111000010111000        |

# การแปลงจากเลข sign-and-magnitude เป็นเลขฐาน 10

การแปลงจากเลข sign-and-magnitude เป็นเลขฐาน 10 มีขั้นตอนดังนี้

ขั้นที่ 1: ไม่ต้องใช้บิตซ้ายสุด (leftmost bit)

ขั้นที่ 2: แปลงเลขฐาน 2 จำนวน  $N-1$  บิตที่เหลือเป็นเลขฐาน 10

ขั้นที่ 3: เติมเครื่องหมาย + หรือ - หน้าเลขฐาน 10 ตามความหมายของ  
บิตซ้ายสุด

## ตัวอย่างที่ 8

จงหาเลขจำนวนเต็มฐานสิบซึ่งเป็นค่าของ 1011011 ที่เก็บ  
ในหน่วยความจำแบบ sign-and-magnitude

### วิธีทำ

ขั้นที่ 1 ไม่ต้องสนใจบิตที่อยู่ซ้ายมือสุด เพราะแทนเครื่องหมายลบเลขที่เหลือ

คือ 0111011

ขั้นที่ 2 เปลี่ยน 0111011 เป็นเลขฐานสิบได้ 59

ขั้นที่ 3 เนื่องจากบิตซ้ายมือสุดเป็น 1 ซึ่งเป็นเลขลบ ผลลัพธ์สุดท้ายคือ -59.

# การประยุกต์ใช้เลข sign-and-magnitude

- \* การเก็บเลขรูปแบบ sign-and-magnitude สำหรับเลขที่มีเครื่องหมายไม่มีการใช้ในคอมพิวเตอร์ในปัจจุบันเนื่องจากเหตุผลหลัก 2 ประการ
  1. การประมวลผลทางคณิตศาสตร์ เช่น + หรือ – กระทำได้ยาก
  2. เนื่องจากมีการแทน 0 อยู่ 2 รูปแบบทำให้โปรแกรมเมอร์เกิดความสับสนในการเขียนโปรแกรม
- \* ข้อดีของการเก็บแบบ sign-and-magnitude เหมาะสมกับการประยุกต์ที่ไม่ใช้การประมวลผลเชิงคณิตศาสตร์ เช่น การเปลี่ยนสัญญาณจาก analog เป็นสัญญาณ digital เป็นต้น

# การเก็บจำนวนเต็มในรูปแบบ one's complement

- รูปแบบการเก็บเลขจำนวนเต็มเป็นเรื่องของการกำหนดรูปแบบและการยอมรับนำไปใช้ จะเห็นว่ามีผู้พยายามคิดและเสนอหลายรูปแบบ
- การแทนในรูปแบบ **one's complement** เป็นการเก็บจำนวนเต็มอิก รูปแบบหนึ่งที่การจัดเก็บเลขจำนวนเต็มบวกจะใช้วิธีการเหมือนกับ **unsigned integer** ส่วนการเก็บจำนวนเต็มลบจะทำการ **complement** เลขจำนวนเต็มบวก เช่น
  - ถ้าต้องการเก็บ +7 ก็จะเก็บเหมือน **unsigned integer**
  - ถ้าต้องการเก็บ -7 จะเก็บเป็น **complement** ของ +7

# การเก็บจำนวนเต็มในรูปแบบ one's complement

- การเก็บจำนวนเต็มในรูปแบบ one's complement ทำได้โดยการเปลี่ยนบิต 0 ให้เป็น 1 และเปลี่ยน 1 ให้เป็น 0 ทั้งหมด
- ถ้า  $N$  เป็นจำนวนบิตที่ใช้แทนจำนวนเต็มแบบ one's complement แล้วค่าของจำนวนที่สามารถเก็บได้จะอยู่ในช่วง  $-(2^{N-1}-1)$  ถึง  $+(2^{N-1}-1)$
- มี 0 อยู่ 2 ตัวในการแทนแบบ one's complement คือ 0 ที่เป็นบวกและ 0 ที่เป็นลบดังนี้ (ใช้ 8 บิต)

$$+0 = 00000000$$

$$-0 = 11111111$$

## ตารางที่ 3.5 พิสัยของจำนวนเต็มที่เก็บแบบ one's complement

| จำนวนบิต | พิสัย          |    |    |                |
|----------|----------------|----|----|----------------|
| <hr/>    |                |    |    |                |
| 8        | -127           | -0 | +0 | +127           |
| 16       | -32767         | -0 | +0 | +32767         |
| 32       | -2,147,483,647 | -0 | +0 | +2,147,483,647 |

# ขั้นตอนการเก็บจำนวนเต็มแบบ one's complement

ขั้นที่ 1: แปลงตัวเลขที่ต้องการเก็บให้เป็นเลขฐาน 2 โดยไม่คำนึงถึงเครื่องหมาย

ขั้นที่ 2: เพิ่ม 0 ทางด้านซ้ายจนครบ N บิต

ขั้นที่ 3: ถ้าเครื่องหมายเป็นบวก ให้เก็บได้เลย แต่ถ้าเครื่องหมายเป็นลบ  
ให้ทำการ complement ทุกบิต (จาก 0 เป็น 1 และจาก 1 เป็น 0)



## Note:

**In one'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.**

## ตัวอย่างที่ 9

จงหาว่าเมื่อจัดเก็บ +7 ในหน่วยความจำขนาด 8-บิตโดยเก็บแบบ  
one's complement

### วิธีทำ

ขั้นที่ 1 เปลี่ยน +7 เป็นเลขฐานสองได้ 111

ขั้นที่ 2 เพิ่ม 0 เข้าไปข้างหน้าจำนวน 5 บิตเพื่อให้ครบ 8 บิตจะได้เท่ากับ  
00000111.

ขั้นที่ 3 เนื่องจากเครื่องหมายเป็นบวก จึงไม่ต้องทำอะไรมาก ผลลัพธ์สุดท้าย

คือ: 00000111

## ตัวอย่างที่ 10

จงหาว่าเมื่อจัดเก็บ -258 ในหน่วยความจำขนาด 16-บิตโดยเก็บแบบ  
one's complement

### วิธีทำ

ขั้นที่ 1 เปลี่ยน +7 เป็นเลขฐานสองได้ 100000010

ขั้นที่ 2 เพิ่ม 0 เข้าไปข้างหน้าจำนวน 7 บิตเพื่อให้ครบ 16 บิตจะได้เท่ากับ  
0000000100000010

ขั้นที่ 3 เนื่องจากเครื่องหมายเป็นลบ จึงต้องทำการ complement แต่ละบิต  
ผลลัพธ์สุดท้ายคือ: 1111111011111101

# ตารางที่ 3.6 ตัวอย่างการจัดเก็บเลขจำนวนเต็มแบบ one's complement ในหน่วยความจำ 2 ขนาด

| จำนวนเต็ม | หน่วยความจำ 8 บิต | หน่วยความจำ 16 บิต |
|-----------|-------------------|--------------------|
| -----     | -----             | -----              |
| +7        | 00000111          | 000000000000111    |
| -7        | 11111000          | 111111111111000    |
| +124      | 01111100          | 000000001111100    |
| -124      | 10000011          | 111111110000011    |
| +24,760   | overflow          | 0110000010111000   |
| -24,760   | overflow          | 1001111101000111   |

# การเปลี่ยนเลข one's complement เป็นเลขฐาน 10

1. ถ้าบิตซ้ายมือสุดเป็น 0 (เป็นเลขบวก) ให้ทำดังนี้

a. แปลงเลขทั้งหมดจาก binary เป็น เลขฐาน 10

b. ใส่เครื่องหมาย + ที่ข้างหน้าของเลขที่ได้

2. ถ้าบิตซ้ายมือสุดเป็น 1 (เป็นเลขลบ) ให้ทำดังนี้

a. เปลี่ยน 0 เป็น 1 และเปลี่ยน 1 เป็น 0 ทั้งหมด

b. แปลงเลขที่ได้จาก a ให้เป็นเลขฐาน 10

c. ใส่เครื่องหมาย - ที่ข้างหน้าของเลขที่ได้

## ตัวอย่างที่ 11

ในหน่วยความจำเก็บค่า 11110110 ซึ่งเป็นจำนวนเต็มที่อยู่ในรูป one's complement จงหาค่าเลขดังกล่าวในรูปฐานสิบ

### วิธีทำ

เนื่องจากบิตซ้ายมือสุดเป็น 1 ดังนั้นจึงเป็นเลขลบ ดำเนินการดังนี้  
ขั้นที่ 1 ทำการ complement จะได้ 00001001

ขั้นที่ 2 ผลที่ได้จากขั้นที่ 1 เมื่อเปลี่ยนเป็นเลขฐานสิบจะได้เท่ากับ 9

ขั้นที่ 3 เนื่องจากเป็นเลขลบ ดังนั้นคำตอบคือ 9



## Note:

**One's complement means reversing all bits. If you one's complement a positive number, you get the corresponding negative number. If you one's complement a negative number, you get the corresponding positive number. If you one's complement a number twice, you get the original number.**

# การประยุกต์เลขแบบ one's complement

- ปัจจุบันไม่การเก็บลักษณะนี้ในคอมพิวเตอร์แล้วด้วยเหตุผลเช่นเดียวกับการเก็บแบบ sign-and-magnitude แต่การเก็บแบบนี้ก็มีประโยชน์มาก เช่นกันคือ
  - เป็นพื้นฐานของการเก็บแบบ two's complement
  - มีคุณสมบัติที่น่าสนใจสำหรับการสื่อสารข้อมูลเช่น การบ่งชี้ข้อผิดพลาดและการแก้ไขข้อผิดพลาดที่เกิดจากการส่งข้อมูล

# การเก็บเลขจำนวนเต็มแบบ two's complement

- การเก็บเลขจำนวนเต็มแบบ one's complement มีปัญหาการแทนที่ทำให้มี 0 สองตัวคือ +0 กับ -0 ซึ่งก่อให้เกิดความสับสนในการคำนวณ เช่นถ้าเราต้องการบวกเลขจำนวนเต็มได้กับ complement ของมัน (+4 กับ -4) จะได้ผลลัพธ์เท่ากับ -0 แทนที่จะเป็น +0
- ถ้า N แทนจำนวนบิตที่ใช้เก็บเลขจำนวนเต็มแบบ two's complement ค่าของตัวเลขที่เก็บได้จะอยู่ระหว่าง  $-(2^{N-1})$  ถึง  $+(2^{N-1}-1)$



Note:

**Two's complement is the most common, the most important, and the most widely used representation of integers today.**

## ตารางที่ 3.7 พิสัยของเลขจำนวนเต็มที่เก็บแบบ two's complement

| จำนวนบิต | พิสัย          |   |                |
|----------|----------------|---|----------------|
| 8        | -128           | 0 | +127           |
| 16       | -32,768        | 0 | +32,767        |
| 32       | -2,147,483,648 | 0 | +2,147,483,647 |

# ขั้นตอนการแทนจำนวนเต็มแบบ two's complement

ขั้นที่ 1: แปลงเลขที่ต้องการแทนเป็นเลขฐาน 2 โดยไม่คำนึงถึงเครื่องหมาย

ขั้นที่ 2: ถ้าเลขฐาน 2 ที่ได้มีน้อยกว่า N บิต ให้เติม 0 ทางซ้ายจนครบ N บิต

ขั้นที่ 3: ถ้าเลขเดิมเป็นบวก ให้จัดเก็บได้เลย แต่ถ้าเป็นลบ ให้พิจารณา  
บิตจากทางขวาสุด ถ้าเป็น 0 ให้คงไว้ แล้วเลื่อนไปทางซ้ายทีละ  
บิตจนกว่าจะพบ 1 หลังจากนี้ให้ทำการ complement บิต  
ทางซ้ายที่เหลือทั้งหมด



Note:

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.

## ตัวอย่างที่ 12

จงหาว่าเมื่อจัดเก็บ +7 ในหน่วยความจำขนาด 8 บิตแบบ two's complement จะมีรูปแบบเป็นอย่างไร?

### วิธีทำ

ขั้นที่ 1 เปลี่ยน +7 ให้เป็นเลขฐานสองได้ 111

ขั้นที่ 2 เพิ่ม 0 จำนวน 5 ตัวเข้าไปข้างหน้าเพื่อให้ครบ 8 บิต ได้ 00000111

ขั้นที่ 3 เนื่องจากเลขเดิมเป็นบวก จึงไม่ต้องทำอะไรต่อไป ผลลัพธ์คือ  
**00000111**

## ตัวอย่างที่ 13

จงจัดเก็บ -40 ในหน่วยความจำขนาด 16-บิตโดยเก็บแบบ two's complement

### วิธีทำ

ขั้นที่ 1 เปลี่ยน 40 เป็นเลขฐานสองได้ 101000.

ขั้นที่ 2 เพิ่ม 0 จำนวน 10 ตัวเข้าไปทางซ้ายเพื่อทำให้ครบ 16 บิตได้

000000000000101000

ขั้นที่ 3 เนื่องจากเลขเดิมเป็นจำนวนเต็มลบ ดังนั้นพิจารณาจากทางขวาเมื่อ

ปล่อยให้เลข 0 จนถึง 1 ตัวแรกคงเดิม แล้วทำการ complement ส่วน

ที่เหลือทั้งหมด ผลลัพธ์คือ: 1111111111011000

## ตารางที่ 3.8 ตัวอย่างการเก็บเลขจำนวนเต็มแบบ two's complement ในหน่วย ความจำ 2 ขนาด

| จำนวนเต็ม | หน่วยความจำขนาด 8 บิต | หน่วยความจำขนาด 16 บิต |
|-----------|-----------------------|------------------------|
| -----     | -----                 | -----                  |
| +7        | 00000111              | 00000000000000111      |
| -7        | 11111001              | 11111111111111001      |
| +124      | 01111100              | 0000000001111100       |
| -124      | 10000100              | 111111110000100        |
| +24,760   | overflow              | 0110000010111000       |
| -24,760   | overflow              | 100111101001000        |

# การเก็บเลขจำนวนเต็มแบบ two's complement

การแปลงจำนวนเต็มแบบ two's complement เป็นเลขฐาน 10

1. ถ้าบิตช้ายสุดเป็น 0 (เลขบวก) ให้ทำดังนี้
  - a. แปลงเลขทั้งหมดจากเลขฐาน 2 เป็นเลขฐาน 10
  - b. ใส่เครื่องหมาย + ข้างหน้าเลขนั้น
2. ถ้าบิตช้ายสุดเป็น 1 (เลขลบ) ให้ทำดังนี้
  - a. ปล่อยให้บิต 0 ทั้งหมดรวมถึงบิต 1 จากทางขวาเหมือนเดิม แล้ว complement ส่วนที่อยู่ทางซ้ายทั้งหมด
  - b. แปลงเลขฐาน 2 ที่ได้เป็นเลขฐาน 10 แล้วใส่เครื่องหมายลบ

จงหาว่าเลขที่เก็บในหน่วยความจำ 11110110 แบบ two's complement แทนเลขฐานสิบจำนวนใด

## วิธีทำ

ขั้นที่ 1 พิจารณาบิตซ้ายมือสุดเป็น 1 และง่วงว่าเป็นเลขลบ

ขั้นที่ 2 ปล่อยให้ 2 บิตขวา มือสุดคือ 10 คงเดิม แล้ว complement ส่วนที่เหลือ

ผลที่ได้คือ 00001010

ขั้นที่ 3 เปลี่ยนค่าที่ได้เป็นเลขฐานสิบได้ 10 แต่เลขเดิมเป็นลบ ผลลัพธ์

สุดท้ายคือ -10



## Note:

**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.**

## ตารางที่ 3.9 สรุปแบบต่างๆของการแทนจำนวนเต็ม

| เลขจำนวนเต็มใน<br>หน่วยความจำ | Unsigned | Sign-and-<br>Magnitude | One's<br>Complement | Two's<br>Complement |
|-------------------------------|----------|------------------------|---------------------|---------------------|
| 0000                          | 0        | +0                     | +0                  | +0                  |
| 0001                          | 1        | +1                     | +1                  | +1                  |
| 0010                          | 2        | +2                     | +2                  | +2                  |
| 0011                          | 3        | +3                     | +3                  | +3                  |
| 0100                          | 4        | +4                     | +4                  | +4                  |
| 0101                          | 5        | +5                     | +5                  | +5                  |
| 0110                          | 6        | +6                     | +6                  | +6                  |
| 0111                          | 7        | +7                     | +7                  | +7                  |
| 1000                          | 8        | -0                     | -7                  | -8                  |
| 1001                          | 9        | -1                     | -6                  | -7                  |
| 1010                          | 10       | -2                     | -5                  | -6                  |
| 1011                          | 11       | -3                     | -4                  | -5                  |
| 1100                          | 12       | -4                     | -3                  | -4                  |
| 1101                          | 13       | -5                     | -2                  | -3                  |
| 1110                          | 14       | -6                     | -1                  | -2                  |
| 1111                          | 15       | -7                     | -0                  | -1                  |

**3.5**

## **EXCESS SYSTEM**

### 3.5.1 Excess System

- Excess system เป็นรูปแบบการเก็บเลขบวกและเลขลบที่ง่ายและสะดวกในการแปลงจากเลขฐาน 10 เป็นเลขฐาน 2 และจากเลขฐาน 2 เป็นเลขฐาน 10 การประยุกต์ที่สำคัญคือการใช้เก็บค่า **exponent** ของเลขเศษส่วนซึ่งจะกล่าวในตอนต่อไป
- ในการแทนแบบ excess system จะมีเลขจำนวนเต็มตัวหนึ่งที่เราใช้ในการบวณการแปลงค่า เลขนี้เรียกว่า **magic number** โดยปกติจะเท่ากับ  $2^{N-1}$  หรือ  $2^{N-1}-1$  เมื่อ N แทนจำนวนบิตที่แทนเลข 1 จำนวน เช่น ถ้า N = 8 แล้วค่า magic number จะเป็น 128 หรือ 127 ซึ่งเราจะเรียกการแทนว่า **Excess-128** หรือ **Excess-127** ตามลำดับ

### 3.5.2 การแทนเลขโดยใช้ Excess System

- การแปลงเลขฐาน 10 ให้เป็นเลขแบบ excess system มีขั้นตอนดังนี้
  - ขั้นที่ 1: บวก magic number เข้ากับตัวเลขที่ต้องการจะแปลง
  - ขั้นที่ 2: แปลงผลลัพธ์ที่ได้เป็นเลขฐาน 2 แล้วเพิ่ม 0 ทางซ้ายจนครบจำนวน N บิต
- การแปลงเลขที่แทนแบบ excess system เป็นเลขฐาน 10 มีขั้นตอนดังนี้
  - ขั้นที่ 1: แปลงเลขในระบบ excess system ให้เป็นเลขฐาน 10
  - ขั้นที่ 2: นำเอาเลข magic number ลบออก ผลที่ได้คือเลขที่ต้องการ

## ตัวอย่างที่ 15

จงแทนเลข -25 ด้วยระบบ Excess\_127 โดยใช้หน่วยความจำขนาด 8 บิต

วิธีทำ

ขั้นที่ 1 บวก 127 กับ -25 ได้ 102

ขั้นที่ 2 เปลี่ยน 102 เป็นเลขฐานสองได้ 1100110

ขั้นที่ 3 เพิ่ม 0 เข้าไปที่บิตซ้ายมือสุดเพื่อให้ครบ 8 บิต ผลลัพธ์สุดท้ายจะเป็น

01100110

## ตัวอย่างที่ 16

จงหาว่าข้อมูลในหน่วยความจำ 11111110 ซึ่งแทนในระบบ Excess\_127 มีค่าเท่ากับเลขฐานสิบจำนวนใด

วิธีทำ

ขั้นที่ 1 เปลี่ยน 11111110 เป็นเลขฐานสิบได้ 254

ขั้นที่ 2 นำ 127 ไปลบออกจาก 254 ผลลัพธ์ที่ได้คือ 127 ซึ่งเป็น

คำตอบที่ต้องการ

**3.6**

## การแทนเลขจำนวนจริง

### 3.6.1 การแทนเลขจำนวนจริง (Floating-point)

- **Floating-point number** คือเลขที่ประกอบด้วยจำนวนเต็มกับส่วนที่เป็นทศนิยม (integer and a fraction) นั่นคือ floating-point เป็นตัวเลขที่แบ่งออกเป็น 2 ส่วนคือส่วนที่เป็นจำนวนเต็มกับส่วนที่เป็นทศนิยม เช่น
  1. ตัวเลข 14.234 มีส่วนจำนวนเต็มเท่ากับ 14 และส่วนที่เป็นทศนิยมเท่ากับ 0.234
  2. ตัวเลข 425.0 มีส่วนจำนวนเต็มเท่ากับ 425 และมีส่วนที่เป็นทศนิยมเท่ากับ 0.0
  3. ตัวเลข 0.325 มีส่วนจำนวนเต็มเป็น 0 และเศษส่วนเป็น 0.325

### 3.6.2 การแปลงเลขจำนวนจริงเป็นเลขฐาน 2

- การแปลงเลข **floating-point** เป็นเลขฐาน 2 มีขั้นตอนดังนี้
  - ขั้นที่ 1 แปลงส่วนที่เป็นจำนวนเต็มให้เป็นเลขฐาน 2
  - ขั้นที่ 2 แปลงส่วนที่เป็นเศษส่วนให้เป็นเลขฐาน 2
  - ขั้นที่ 3 ใส่จุดทดแทนระหว่างทั้งสองส่วนจากขั้นที่ 1 และขั้นที่ 2

**หมายเหตุ:** การแปลงในขั้นที่ 1 ใช้วิธีการจากบทที่ 2 ส่วนการแปลงในขั้นที่ 2 มีรายละเอียดดังต่อไปนี้

# การแปลงส่วนที่เป็นเลขทศนิยมเป็นเลขฐาน 2

- การแปลงส่วนที่เป็นเลขทศนิยมเป็นเลขฐาน 2 โดยใช้วิธีการคูณซ้ำๆ (repetitive multiplication) จนกระทั่งส่วนที่เป็นทศนิยมเป็น 0
- ตัวอย่าง: จงแปลง 0.125 เป็นเลขฐาน 2

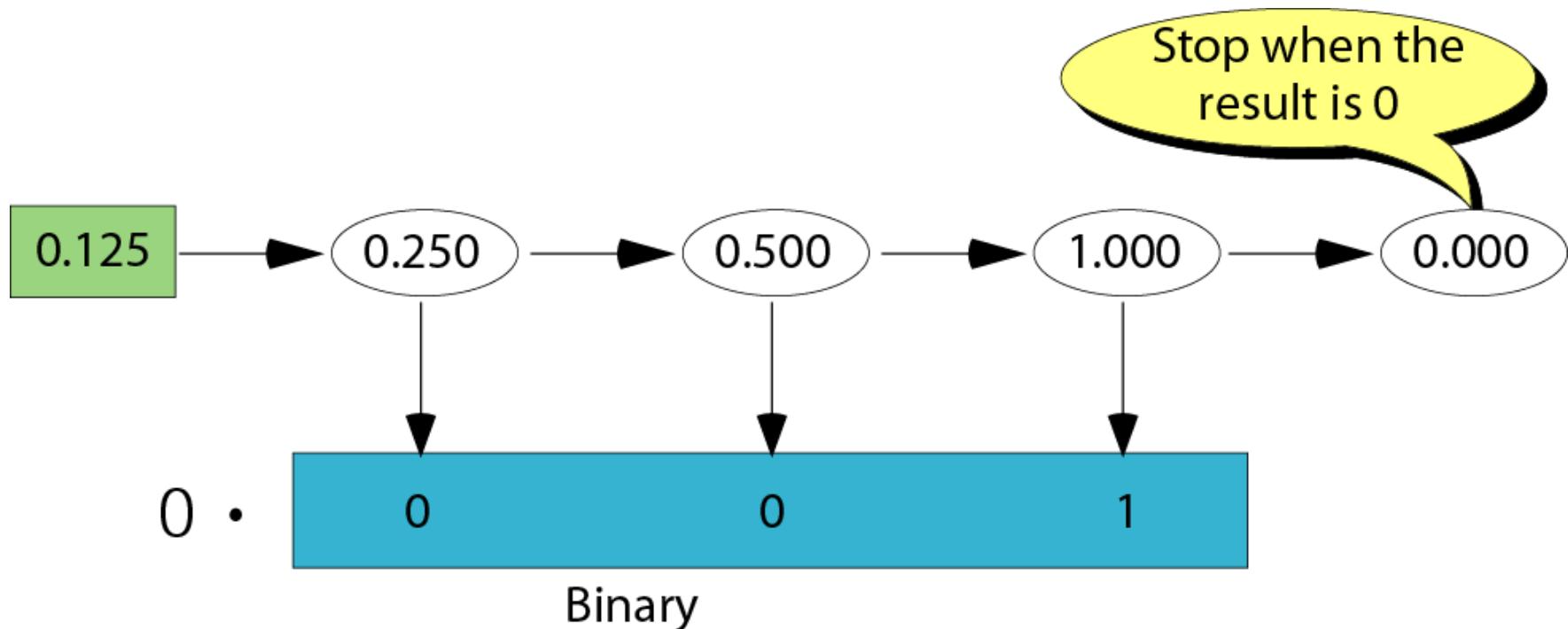
ขั้นที่ 1: คูณ 0.125 ด้วย 2 ได้ 0.250 มีส่วนจำนวนเต็ม = 0

ขั้นที่ 2: คูณ 0.250 ด้วย 2 ได้ 0.500 มีส่วนจำนวนเต็ม = 0

ขั้นที่ 3: คูณ 0.500 ด้วย 2 ได้ 1.000 มีส่วนจำนวนเต็ม = 1

ดังนั้นจะได้ค่าตอบ = 0.001 (ดังรูปที่ 3.7)

### รูปที่ 3-7 การเปลี่ยนส่วนที่เป็นเลขทศนิยมเป็นเลขฐานสอง



## ตัวอย่างที่ 17

จงเปลี่ยนเลขทศนิยม 0.875 ให้เป็นเลขฐานสอง

### วิธีทำ

ขั้นที่ 1 เขียนเลขทศนิยมที่ต้องการเปลี่ยนไว้ทางซ้ายมือสุด

ขั้นที่ 2 คูณเลขทศนิยมด้วย 2 นำส่วนที่เป็นจำนวนเต็มจากผลลัพธ์ไปเป็น

binary digit

ขั้นที่ 3 ทำซ้ำขั้นที่ 2 จนกระทั้งผลลัพธ์เป็น 0.0

$$0.875 \rightarrow 1.750 \rightarrow 1.5 \rightarrow 1.0 \rightarrow 0.0$$

0 . 1 1 1

## ตัวอย่างที่ 18

จงเปลี่ยนเลขทศนิยม 0.4 เป็นเลขฐานสองจำนวน 6 บิต

### วิธีทำ

ขั้นที่ 1 เขียนเลขทศนิยมที่ต้องการเปลี่ยนไว้ทางซ้ายมือสุด

ขั้นที่ 2 คูณเลขทศนิยมด้วย 2 นำส่วนที่เป็นจำนวนเต็มจากผลลัพธ์ไปเป็น binary digit

ขั้นที่ 3 ทำซ้ำขั้นที่ 2 จนกระทั่งผลลัพธ์เป็น 0.0 แต่ในกรณีนี้จะไม่มีโอกาสเป็น 0.0 จึงหยุดเมื่อได้ binary digit ครบ 6 บิต

0.4 → 0.8 → 1.6 → 1.2 → 0.4 → 0.8 → 1.6  
0 . 0 1 1 0 0 1

### 3.6.3 การทำ Normalization

- พิจารณาการเก็บเลข 71.3125 หรือ +1000111.0101 เราจะต้องเก็บเครื่องหมาย เก็บบิตทั้งหมด และเก็บตำแหน่งของจุดทศนิยมไว้ในหน่วยความจำ แม้ว่าการเก็บแบบนี้จะทำได้ แต่การกระทำทางคณิตศาสตร์เกี่ยวกับตัวเลขที่เก็บแบบนี้จะมีความยุ่งยากมาก จึงจำเป็นจะต้องหาวิธีการที่เป็นมาตรฐานสำหรับแทนเลข floating-point
- คำตอนคือการทำ Normalization เป็นการเลื่อนจุดทศนิยมไปทางซ้ายหรือขวาจนกระทั่งได้ 1 เป็นจำนวนเต็มเพียงตัวเดียวที่อยู่ทางซ้ายของจุดทศนิยมดังนี้: 1.xxxxxxxxxxxxxxx

# การทำ Normalization (ต่อ)

- เพื่อให้ค่าที่ได้ยังคงเดิม เราจะทำการคูณเลขที่อยู่ในรูปมาตรฐานด้วย  $2^e$  เมื่อ e คือจำนวนบิตที่จุดทศนิยมเลื่อนไป ถ้าเลื่อนไปทางซ้ายค่า e จะเป็นบวก ถ้าเลื่อนไปทางขวาค่า e จะเป็นลบ
- จากนั้นครื่องหมายบวกหรือลบจะถูกเพิ่มเข้าไปข้างหน้าตัวเลขขึ้นอยู่กับค่าเดิมของเลขตัวนั้น เช่น $+1010001.11001$  เลื่อนไปทางซ้าย 6 ตำแหน่งได้  $+2^6 \times 1.01000111001$  $-0.001110011$  เลื่อนไปทางขวา 3 ตำแหน่งได้  $-2^{-3} \times 1.110011$

**Table 3.10 ตัวอย่างของการทำ normalization**

| Original Number  | Move | Normalized                  |
|------------------|------|-----------------------------|
| +1010001.1101    | ← 6  | $+2^6 \times 1.01000111001$ |
| -111.000011      | ← 2  | $-2^2 \times 1.11000011$    |
| - +0.00000111001 | 6 →  | $+2^{-6} \times 1.11001$    |
| -0.001110011     | 3 →  | $-2^{-3} \times 1.110011$   |

# เครื่องหมาย (sign), ค่าเลขยกกำลัง (exponent) และค่า曼นทิชชา (mantissa)

- หลังจากการทำ normalization กับตัวเลขแล้ว เราจะเก็บข้อมูลเกี่ยวกับตัวเลขเพียง 3 ส่วนคือ เครื่องหมาย ค่าเลขยกกำลัง และค่า曼นทิชชา เช่น

$+2^6 \times 1.01000111001$

เครื่องหมาย คือ

+ (เก็บเป็น 0=บวก หรือ 1=ลบ)

ค่าเลขยกกำลัง คือ

6 (กำลังของ 2 อาจเป็นบวก, ลบ)

ค่า曼นทิชชา คือ

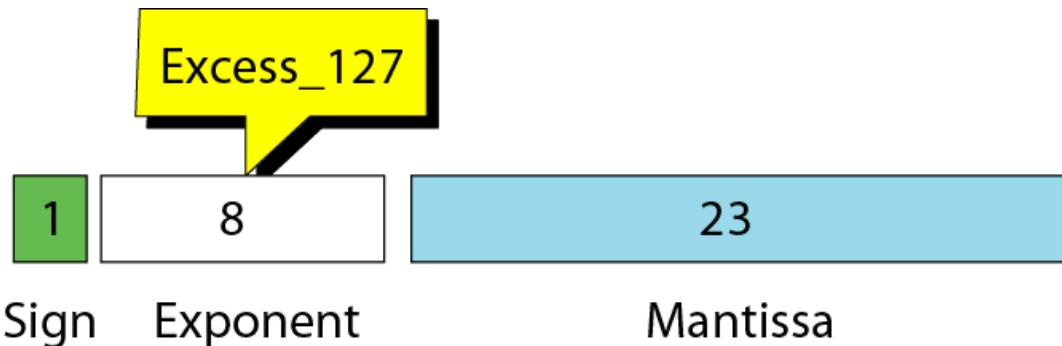
01000111001 (unsigned int.)

### 3.6.4 มาตรฐาน IEEE

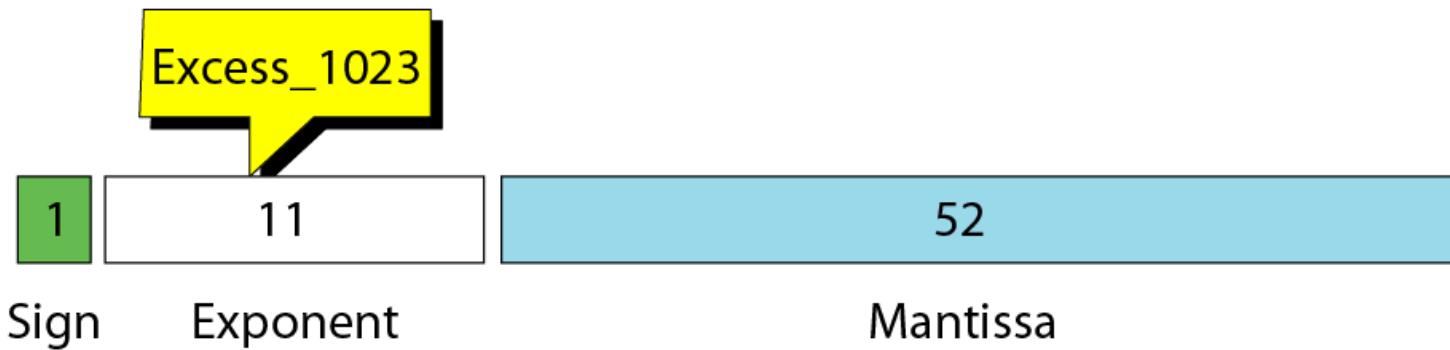
- IEEE ย่อมาจาก **The Institute of Electrical and Electronics Engineers** หรือสถาบันวิศวกรรมไฟฟ้าและอิเล็กทรอนิกส์ ได้กำหนด มาตรฐานการเก็บไว้ 2 แบบเพื่อใช้เก็บเลข floating-point คือ **single precision** และ **double precision** ดังรูปที่ 3.8

## รูปที่ 3-8

### มาตรฐาน IEEE สำหรับการแทนเลขจำนวนจริง



a. Single Precision



b. Double Precision

# การแทนเลข floating-point แบบ single precision

- การเก็บเลข floating-point ที่ normalized แล้วแบบ single precision:
  - ขั้นที่ 1: เก็บเครื่องหมายของเลขเป็น 0 (บวก) และ 1 (ลบ)
  - ขั้นที่ 2: เก็บค่าเลขยกกำลัง (กำลังของ 2) แบบ Excess-127
  - ขั้นที่ 3: เก็บค่าแม่นทิชชาแบบ unsigned integer

## ตัวอย่างที่ 19

จงหาว่าเลขที่แทนด้วยรูปแบบ normalization ต่อไปนี้จะถูกหีบใน  
หน่วยความจำแบบใด:  $+ 2^6 \times 1.01000111001$

### วิธีทำ

ขั้นที่ 1 เนื่องจากเครื่องหมายเป็นบวก เมื่อแทนด้วยระบบ Excess\_127 แล้ว

ค่า exponent จะเท่ากับ 133

ขั้นที่ 2 เพิ่ม 0 เข้าไปทางขวาจนครบ 23 บิต

ขั้นที่ 3 ตัวเลขที่จะเก็บในหน่วยความจำคือ:

0 10000101 010001110010000000000000

## ตารางที่ 3.11 ตัวอย่างการแทนเลข floating-point

| เลขที่จะเก็บ              | +/- | เลขยกกำลัง | แมนพิชช่า                        |
|---------------------------|-----|------------|----------------------------------|
| $-2^2 \times 1.11000011$  | 1   | 10000001   | 11000011000000000000000000000000 |
| $+2^{-6} \times 1.11001$  | 0   | 01111001   | 11001000000000000000000000000000 |
| $-2^{-3} \times 1.110011$ | 1   | 01111100   | 11001100000000000000000000000000 |

# การแปลงเลข floating-point ที่เก็บในหน่วยความจำเป็นเลขฐาน 10

ขั้นที่ 1: ใช้บิตซ้ายมือสุดแทนเครื่องหมาย 0 = บวก และ 1 = ลบ

ขั้นที่ 2: แปลง 8 บิตถัดไปเป็นเลขฐาน 10 แล้วนำ 127 ไปลบออก ผลลัพธ์ที่ได้เป็นค่าเลขยกกำลัง (exponent)

ขั้นที่ 3: เพิ่ม 1 และจุดทศนิยมใน 23 บิตถัดไป (ไม่ต้องสนใจ 0's ที่อยู่ทางขวา)

ขั้นที่ 4: เลื่อนจุดทศนิยมไปอยู่ณ ตำแหน่งที่ถูกต้องโดยดูจากค่าเลขที่ยกกำลัง (exponent)

ขั้นที่ 5: แปลงเลขทั้งหมดที่ได้เป็นเลขฐาน 10

ขั้นที่ 6: แปลงส่วนที่เป็นทศนิยมให้เป็นเลขฐาน 10

ขั้นที่ 7: รวมทั้งสองส่วน (5 และ 6) เข้าด้วยกัน

## ตัวอย่างที่ 20

จงแปลงเลข floating-point 32 บิตต่อไปนี้ให้อยู่ในรูป normalization

1 01111100 11001100000000000000000000

วิธีทำ

ขั้นที่ 1 เนื่องจากบิตช้ายมือสุดเป็น 1 ดังนั้นเป็นเลขลบ

ขั้นที่ 2 ค่าเลขยกกำลังคือ  $-3$  ( $124 - 127$ )

ขั้นที่ 3 ตัวเลขหลังจากทำ normalization คือ

$$-2^{-3} \times 1.110011$$

**3.7**

## รูปแบบเลขฐานสิบหก

# เลขฐาน 16

- ข้อมูลที่เป็นตัวเลขสามารถแทนด้วยเลขฐาน 16 ตัวอย่างเช่น 48 สามารถแทนด้วยหน่วยความจำที่ใช้ 8-bit pattern แบบ unsigned format เป็น 00110000 หรือแทนด้วยเลขฐาน 16 เป็น x30
- ในลักษณะที่คล้ายคลึงกัน ตัวเลขอย่างเช่น 81.5625 สามารถแทนโดยใช้มาตรฐาน IEEE เป็น 01000010101000111001000000000000 หรือแทนเป็นเลขฐาน 16 ได้เป็น x42A39000

# คำสำคัญของบทที่ 3

**binary system**

**decimal system**

**double precision**

**excess system**

**floating-point number**

**fraction**

**mantissa**

**normalization**

**one's complement**

**sign-and-magnitude**

**single-precision format**

**two's complement**

**unsigned integer**

**whole number**