การแสดงข้อมูลในคอมพิวเตอร์ (Data Representation)

1 ระบบตัวเลข (Number System)

สำหรับคนทั่วไป ระบบตัวเลขที่ใช้กันทั่วไปในชีวิตประจำวัน คือ ระบบเลขฐานสิบ (Decimal Number System) ซึ่งประกอบด้วยเลข 0,1,2,3,4,5,6,7,8,9 หรือพูดง่าย ๆ คือ แต่ละหลักจะมีค่าได้สูงสุดไม่เกิน 9 ยกตัวอย่าง เช่น เลขสูงสุดในหลักหน่วยคือ เลข 9 เลขจำนวนเต็มสูงสุดที่น้อยกว่า 1000 คือ 999 เป็นต้น

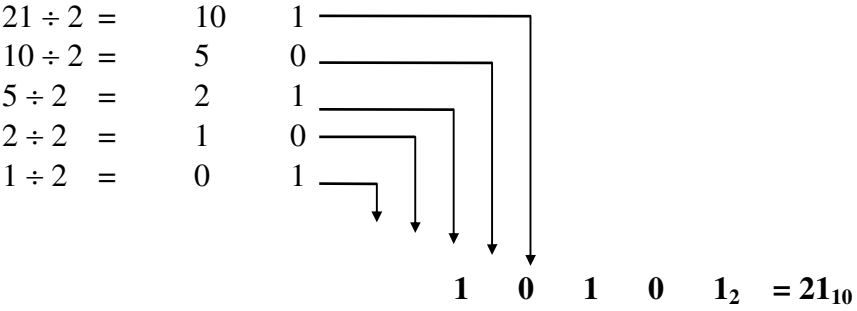
ระบบเลขฐานสิบ จะง่ายสำหรับการคำนวณและบันทึกค่าของตัวเลขต่าง ๆ ที่ใช้ในสังคมของมนุษย์ หรืออาจจะกล่าวได้ว่า ระบบเลขฐานสิบเป็นส่วนหนึ่งของภาษาคน ก็ได้ แต่ในคอมพิวเตอร์ จะใช้เลขฐานสอง (Binary Number System) ในการแสดงข้อมูลและคำสั่งต่าง ๆ ซึ่งหมายถึง ใช้เลขฐานสองแทนภาษานั่นเอง ดังนั้น ในการแสดงข้อมูลที่ใช้บนเครื่องคอมพิวเตอร์ ระบบตัวเลขที่ใช้กันมาก ได้แก่

1. **ระบบเลขฐานสอง** (Binary Number System) ประกอบด้วยเลข 0,1 โดย 1 หลักในเลขฐานสองจะเรียกว่า 1 bit
2. **ระบบเลขฐานแปด** (Octal Number System) ประกอบด้วยเลข 0,1,2,3,4,5,6,7
3. **ระบบเลขฐานสิบหก** (Hexadecimal Number System) ประกอบด้วยเลข 0,1,2,3,4,5,6,7,8,9 สำหรับแทนจำนวนที่น้อยกว่า 10 และตัวอักษร A,B,C,D,E,F สำหรับแทนเลข 10,11,12,13,14,15

อนึ่ง ในการเขียนโปรแกรม และแสดงข้อมูลที่เกี่ยวข้องกับคอมพิวเตอร์ เลขฐานสิบหก มักจะแสดงในรูปที่นำหน้าด้วย 0x เช่น $(AB)_{16}$ จะแสดงด้วย 0xAB เป็นต้น

ระบบเลขฐานแปด และระบบเลขฐานสิบหก ใช้ในการแสดงข้อมูลที่เป็นเลขฐานสอง เพื่อให้สามารถอ่านได้รวดเร็ว และแม่นยำยิ่งขึ้น เพราะในคอมพิวเตอร์ จะใช้เลขฐานสองที่มีจำนวนหลักมาก (เช่น 8 bits, 16 bits, 32 bits) ถ้าแสดงอยู่ในรูปเลขฐานสอง จะทำให้ลำบากในการอ่าน และอาจเกิดการผิดพลาดในการอ่านได้ จึงนิยมใช้เลขฐานแปด หรือเลขฐานสิบหกแทนการใช้เลขฐานสอง โดยเฉพาะเลขฐานสิบหก นิยมใช้กันมาก เพราะความยาวของข้อมูลที่ใช้ในคอมพิวเตอร์ ส่วนใหญ่จะเป็นผลคูณของเลข 4 (e.g. 8 bits, 16 bits etc.)

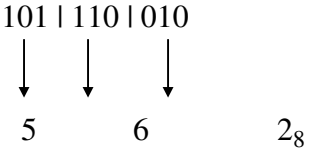
ตัวอย่างวิธีการแปลงเลขจำนวนเต็มฐานสิบให้เป็นเลขฐานสอง



วิธีการแปลงเลขฐานสอง เป็นเลขฐานแปด

แปดเท่ากับ 2 ยกกำลัง 3 ดังนั้น เลขฐานสอง 3 หลักจึงเท่ากับเลขฐานแปด 1 หลัก เช่น $011_2 = 3_8$ ดังนั้น เราสามารถทำการแปลงเลขฐานสองเป็นเลขฐานแปด โดยการแปลงเลขฐานสองให้เป็นเลขฐานแปดทีละ 3 หลัก

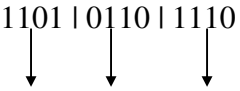
ตัวอย่างที่ 1: 1011100010_2



วิธีการแปลงเลขฐานสอง เป็นเลขฐานสิบหก

สิบหกเท่ากับ 2 ยกกำลัง 4 ดังนั้น เหมือนกับกรณีเลขฐานแปด กล่าวคือ เลขฐานสอง 4 หลักจึงเท่ากับเลขฐานแปด 1 หลัก เช่น $1011_2 = B_8$ ดังนั้น เราสามารถทำการแปลงเลขฐานสองเป็นเลขฐานแปด โดยการแปลงเลขฐานสองให้เป็นเลขฐานสิบหกทีละ 4 หลัก

ตัวอย่างที่ 2 : 110101101110_2



2 การแสดงข้อมูลในคอมพิวเตอร์ (Data Representation)

ข้อมูลในคอมพิวเตอร์ทุกตัวจะถูกแสดงโดยใช้เลขฐานสอง แต่จะต่างกันที่จำนวน bit ที่ใช้ในการแสดง หรืออีกนัยหนึ่งคือ ความยาวของแต่ละข้อมูลจะต่างกัน โดยความยาวนี้ จะใช้หน่วยเป็น byte ซึ่ง 1 byte จะเท่ากับ 8 bits หรือเลขฐานสองจำนวนแปดหลัก

หมายเหตุ :

1. byte ที่ว่านี้ จะใช้เป็นหน่วยความจำด้วย โดยหน่วยความจำ 1 byte จะหมายความว่า สามารถบันทึก (หรือจำ) ข้อมูลที่มีความยาว 1 byte ได้ 1 ตัว เพราะฉะนั้น ในกรณีที่ใช้เลขฐานสิบในการอ้างอิง หน่วยความจำ 1 MB (Megabytes) จะหมายความว่า หน่วยความจำนั้นสามารถบันทึกข้อมูลที่มีความยาว 1 byte ได้ 1 ล้านตัว
2. ในกรณีของหน่วยความจำและหน่วยความจำชั่วคราว เนื่องจากใช้เลขฐานสองในการอ้างอิง จึงมีการกำหนดมาตรฐานของขนาดหน่วยความจำ โดยเปรียบเทียบกับเลขยกกำลังของ 2 ดังต่อไปนี้

UNIT	SYMBOL	POWER of 2	VALUE
Byte		2^0	1
Kilobyte	KB	2^{10}	1,024
Megabyte	MB	2^{20}	1,048,576
Gigabyte	GB	2^{30}	1,073,741,824
Terabyte	TB	2^{40}	1,099,511,627,776

ถ้าใช้วิธีการคิดแบบนี้ 64 KB จะเท่ากับ 2^{16} = 65536 Bytes ระบบปฏิบัติการ Windows จะแสดงขนาดของหน่วยความจำโดยใช้การคำนวณข้างต้น อันเป็นผลให้ 400 GB อาจได้รับการแสดงเป็น 372 GB เป็นต้น

ข้อมูลที่ใช้ในคอมพิวเตอร์ จำแนกได้เป็นประเภทใหญ่ ๆ ดังนี้

2.1 ตัวอักษร (ตัวอักขระ, Character) ประกอบด้วย ตัวอักษรธรรมดาที่ใช้กันทั่วไป เช่น A, B, a, b etc. ตัวเลข เช่น 1, 2, ... สัญลักษณ์ต่าง ๆ เช่น #, *, % etc. และตัวอักษรที่มีความหมายพิเศษต่าง ๆ เช่น TAB, SPACE, BACKSPACE, DEL, ENTER(NEW LINE) เป็นต้น ซึ่งใช้คำเรียกว่า Special Character ในคอมพิวเตอร์ จะแสดงตัวอักษรเหล่านี้ด้วยเลขฐานสอง 8 หลัก หรือ 1 byte

รหัสตัวอักษรที่ใช้บนคอมพิวเตอร์มีหลายแบบ ได้แก่

- **BCD (Binary Coded Decimal)** เป็นระบบตัวอักษรที่ใช้ในคอมพิวเตอร์รุ่นแรก ๆ สร้างโดย IBM มีความยาว 6 bit จึงสามารถแสดงตัวอักษรได้เพียง 64 ตัวอักษร และมีเพียงตัวพิมพ์ใหญ่เท่านั้น
- **EBCDIC (Extended Binary Coded Decimal Interchange Code)** เป็น Code ที่ IBM ปรับปรุงจาก BCD เพื่อให้สามารถแสดงตัวอักษรได้มากขึ้น EBCDIC มีขนาด 8 bit และใช้แต่ในเฉพาะเครื่องประเภท Minicomputer และ Mainframe ไม่มีการนำมาใช้ในเครื่อง PC
- **ASCII (American Standard Code for Information Interchange)** เป็น Code ที่ถูกสร้างขึ้นโดย ANSI (American National Standards Institute) เพื่อใช้เป็นมาตรฐานของข้อมูล มีความยาว 8 bit แต่ใช้จริงแค่ 7 bit

โดย bit ที่ 8 ใช้เป็น Parity Bit สำหรับตรวจสอบข้อผิดพลาดของข้อมูล แต่เนื่องจากในปัจจุบัน ความสำคัญของ Parity Bit ได้ลดน้อยลงไป IBM จึงได้ทำการปรับปรุง ASCII ให้สามารถแสดงได้ 256 ตัวอักษร

ตารางที่ 1 ตัวอย่างรหัสตัวอักษร ASCII

ตัวอักษร	เลขฐานสิบหกที่ใช้ในการแสดง
A	0x41
B	0x42
a	0x61
#	0x23
*	0x2A
SPACE	0x20
^H (Backspace)	0x08

2.2 เลขจำนวนเต็ม (integer) ในคอมพิวเตอร์ โดยปรกติ จะใช้ข้อมูล 2 หรือ 4 bytes ในการแสดงเลขจำนวนเต็ม ใน Microprocessor ที่ใช้ใน PC จะใช้ 2 bytes ในการแสดงเลขจำนวนเต็ม ดังนั้น ถ้าคิดเฉพาะกรณีเลขจำนวนเต็มบวกอย่างเดียว (unsigned int : เลขจำนวนเต็มที่ไม่มีสัญลักษณ์บวกลบ) เลขจำนวนเต็มที่สามารถแสดงได้จะอยู่ในระหว่าง 0~65535 (2^{16}-1) แต่ถ้าหากเป็นจำนวนเต็มที่มีสัญลักษณ์ เลขที่แสดงได้ จะอยู่ในระหว่าง -32768 (2^{15}) ~ 32767 (2^{15}-1) การแสดงเลขจำนวนเต็มลบในเลขฐานสอง มีหลายวิธี เช่น ใช้ bit ซ้ายสุด (Most Significant Bit; MSB) แสดงสัญลักษณ์บวกลบ เป็นต้น วิธีที่นิยมมากที่สุดในปัจจุบัน คือ การใช้ 2's complement แสดงเลขจำนวนเต็มลบ เพราะง่ายต่อการคำนวณ

q's complement: ถ้ากำหนดให้ $N = a_n a_{n-1} \cdots a_1 a_{0_q}$ โดยให้ q แสดงเลขฐาน แล้ว q's complement ของ N, M_q จะเท่ากับ

$$M_q = q^{n+1} - N \tag{1}$$

และ (q-1)'s complement ของ N, M_{q-1} จะเท่ากับ

$$M_{q-1} = q^{n+1} - q^0 - N \tag{2}$$

ซึ่งหมายความว่า

$$M_q = M_{q-1} + q^0 \tag{3}$$

หรือกล่าวโดยสรุปแล้วก็คือ (q-1)'s complement ของ N จะเท่ากับ q's complement ของ N ลบด้วย 1 (เพราะ q^0=1)

<u>หมายเหตุ</u> : ผลบวกของ M_q กับ N จะเท่ากับ q^{n+1}

<u>ตัวอย่างที่ 3</u>: 10's complement ของ 3 = 10 - 3 = 7
9's complement ของ 3 = 10 - 1 - 3 = 6
16's complement ของ $(7A)_{16} = 16^2 - 122 = 134$
15's complement ของ $(7A)_{16} = 16^2 - 1 - 122 = 133$

<u>ตัวอย่างที่ 4</u> : 2's complement และ 1's complement ของเลขฐานสอง

2's complement ของ 0110_2 $= 10000_2 - 0110_2$
$= 16 - 6 = 10$
$= 1010_2$

1's complement ของ 0110_2 $= 10000_2 - 1_2 - 0110_2$
$= 16 - 1 - 6 = 9$
$= 1001_2$

จากตัวอย่างที่ 2 จะสังเกตได้ว่า 1's complement ของเลขฐานสอง หาได้โดยการเปลี่ยนเลขในแต่ละหลักของเลขฐานสองจาก 1 เป็น 0 และจาก 0 เป็น 1 เช่น 1's complement ของ 10010001_2 ก็จะเท่ากับ 01101110_2 และดังได้กล่าวไว้ข้างต้น 2's complement ของ N เท่ากับ 1's complement ของ N บวกด้วย 1 เพราะฉะนั้น วิธีหา 2's complement ที่ง่ายที่สุด คือ หา 1's complement แล้วบวกด้วย 1

ในคอมพิวเตอร์ทั่วไปในปัจจุบัน จะใช้ 2's complement ที่กล่าวข้างต้นในการแสดงจำนวนเต็มลบ โดยจะมีหลักการง่าย ๆ คือ ถ้าคิดเฉพาะจำนวน bit ที่มีอยู่ 2's complement บวกกับ N จะมีค่าเท่ากับ 0

ตัวอย่าง : ในกรณีเลข 4 bit ให้ $N = 0101_2$ แล้วจะได้ 2's complement ของ N, $M_2 = 1011_2$ ดังนั้น $N+M_2= 10000_2$ คิดเฉพาะ 4 bit จะทำให้ได้ผลลัพธ์เท่ากับ 0 (สรุปแล้ว จะได้ 5 + (-5) = 0)

ตัวอย่างที่ 5: กรณีเลข 8 bit จะได้ตามตารางต่อไปนี้

ตารางที่ 2 การแสดงเลขจำนวนเต็มลบ

จำนวนเต็มบวก	เลขฐานสอง	จำนวนเต็มลบ	เลขฐานสอง (2's complement)
0	00000000		
1	00000001	-1	11111111
50	00110010	-50	11001110
100	01100100	-100	10011100
127	01111111	-127	10000001
		-128	10000000

หมายเหตุ ในการลบเลขในคอมพิวเตอร์ จะแปลงเลขที่จะลบให้อยู่ในรูป 2's complement แล้วทำการบวกกัน ซึ่งทำให้สามารถใช้วงจรที่ใช้ในการคำนวณผลบวก คำนวณผลลบได้ โดยไม่ต้องมีวงจรเพิ่มเติม เช่น $N_1 - N_2 = N_1 +$ 2's complement of N_2

ข้อจำกัดของเลขจำนวนเต็ม ดังที่ได้กล่าวไว้ข้างต้น กรณีที่มีแต่เลขจำนวนเต็มบวก หรือ Unsigned Integer จะแสดงเลขได้ตั้งแต่ $0 \sim 2^{n-1}$ โดย n แสดงจำนวน bit แต่เมื่อต้องมีการแสดงเลขจำนวนเต็มลบด้วย หรือ Signed Integer เลขจำนวนเต็มที่แสดงได้จะเปลี่ยนไป โดยค่าจำนวนเต็มบวกสูงสุดที่แสดงได้ จะเท่ากับ $01111...111_2$ จำนวนของเลข 1 ในที่นี้ จะเท่ากับ จำนวน bit ลบด้วย 1 กล่าวคือ ถ้าเป็นเลข 8 bit ก็จะเท่ากับ 01111111_2 (= 127) ถ้าเป็นเลข 16 bit ก็จะเท่ากับ $0111111111111111_2 = 7FFF_{16} = 32767$ ส่วนค่าจำนวนลบต่ำสุดที่แสดงได้จะเท่ากับ $1000...000_2 = -2^{n-1}$ โดยให้ n แสดงจำนวน bit เพราะฉะนั้น เลข 8 bit จะแสดงเลขจำนวนลบต่ำสุดได้ -128 เลข 16 bit จะแสดงได้ -32768 ความสัมพันธ์ระหว่างจำนวน bit กับเลขจำนวนเต็มที่แสดงได้จะสรุปได้ตามตารางดังต่อไปนี้

ตารางที่ 3 ข้อจำกัดของเลขจำนวนเต็ม

จำนวน bit	เลขจำนวนเต็มที่แสดงได้	
	Unsigned	**Signed**
8	0 ~ 255	-128 ~ 127
16	0 ~ 65535	-32768 ~ 32767
32	0 ~ 4294967295	-2147483648 ~ 2147483647

ข้อสังเกต การคำนวณเลขจำนวนเต็มจะได้ค่าถูกต้องก็ต่อเมื่อผลลัพธ์จากการคำนวณอยู่ในช่วงที่สามารถแสดงได้

ตัวอย่างที่ 6 : กรณีเลข 8 bit 64 + 63 ได้ผลลัพธ์ที่ถูกต้อง แต่ 64 + 64, -65 + (-64) จะไม่ได้ผลลัพธ์ที่ถูกต้อง

สาเหตุ : $01000000_2 + 00111111_2 = 01111111_2$

$01000000_2 + 01000000_2 = 10000000_2 = -128$ -> Not Correct

$11000000_2 + 10111111_2 = 01111111_2 = 127$ -> Not Correct

กรณีที่ตัวเลขที่จำเป็นต้องแสดงไม่อยู่ในช่วงที่เครื่องคอมพิวเตอร์สามารถแสดงได้นั้น (e.g. 64+64, -65-64) เราเรียกว่า *Overflow* ซึ่งเป็นข้อผิดพลาดที่มักเกิดขึ้นบ่อย โดยเฉพาะเมื่อเกิดการหารด้วย 0 ขึ้น

2.3 เลขทศนิยม วิธีแสดงเลขทศนิยมในคอมพิวเตอร์ มีอยู่ 2 วิธีหลัก ๆ คือ Fixed-Point Representation และ Floating-Point Representation ในคอมพิวเตอร์ยุคปัจจุบัน โดยมากจะใช้แบบหลัง เนื่องจากสะดวกในการใช้คำนวณและแสดงค่าได้มากกว่า

2.3.1 **Fixed-Point Representation** เป็นวิธีที่ใช้กันทั่วไปในคอมพิวเตอร์ยุคแรก ๆ จะใช้วิธีกำหนดตำแหน่งของเลขทศนิยมลงไปอย่างตายตัว ซึ่งโดยทั่วไป จะกำหนดไว้ที่ด้านซ้ายของ bit ซ้ายสุด (Most Significant Bit : MSB) และ ด้านขวาของ bit ขวาสุด (Least Significant Bit : LSB)

กรณี 1 : ด้านซ้ายของ MSB แสดงเลขหลังจุดทศนิยม

.10000000 = 0.50

.01010000 = 0.3125

กรณี 2 : ด้านขวาของ LSB แสดงส่วนที่เป็นจำนวนเต็ม

00001000**.** = 8

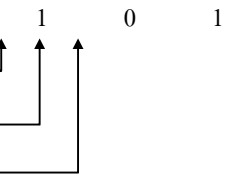
00100100**.** = 36

ดังนั้น ในการคำนวณเลขที่แสดงด้วยวิธีนี้ จะมีข้อจำกัดมาก วิธีการแสดงเลขทศนิยมแบบนี้มีข้อดี คือ ออกแบบวงจรที่ใช้ในการคำนวณได้ง่าย แต่จะมีข้อเสีย คือ การเขียนโปรแกรมเพื่อใช้งานจะซับซ้อน และประสิทธิภาพที่ได้จะไม่ดี อีกประการหนึ่ง คือ จะไม่สามารถแสดงเลขทศนิยมจำนวนมากได้

<u>การแปลงเลขทศนิยมให้เป็นเลขฐานสอง</u> วิธีที่ใช้ทั่วไปในการแปลงทำได้ดังตัวอย่าง ดังต่อไปนี้

<u>ตัวอย่าง 7</u> : 0.625

Product	Integral Part
0.625 × 2 = 1.25	1
0.25 × 2 = 0.50	0
0.50 × 2 = 1.00	1
0.0	



$\therefore$ $0.625 = .101_2$

<u>ตัวอย่าง 8</u> : 0.81

Product	Integral Part
0.81 × 2 = 1.62	1
0.62 × 2 = 1.24	1
0.24 × 2 = 0.48	0
0.48 × 2 = 0.96	0
0.96 × 2 = 1.92	1
0.92 × 2 = 1.84	1
0.84 ...	

$\therefore$ $0.81 = .110011_2$ (โดยประมาณ)

หมายเหตุ ในตัวอย่าง 7 จะสามารถหารจนได้เศษเป็น 0 ซึ่งหมายความว่า เป็นเลขหารลงตัว อันทำให้สามารถแสดงด้วยเลขฐานสองได้อย่างเที่ยงตรง แต่ในตัวอย่าง 8 ไม่สามารถหารลงตัวได้ กล่าวคือ จะไม่สามารถแปลงเป็นเลขฐาน 2 ให้ตรงกับตัวเลขฐานสิบได้ ซึ่งข้อผิดพลาดนี้ เรียกว่า *Round-Off Error*

2.3.2 **Floating-Point Representation** เป็นวิธีที่ใช้กันทั่วไปในคอมพิวเตอร์ยุคปัจจุบัน (จะสังเกตได้จากชื่อเรียกของ Processor ที่ใช้ในการคำนวณเลขทศนิยมว่า Floating Point Unit : FPU) มีหลายรูปแบบ แต่รูปแบบที่เป็นมาตรฐาน จะจัดตัวเลขทศนิยมที่ต้องการแสดง N ให้อยู่ในรูป

$$N = \pm M \times 2^{\pm E} \tag{4}$$

โดย M มีชื่อเรียกว่า Significand (หรือ Mantissa, Fraction) และ E แทนเลขชี้กำลัง (Exponent) และแบ่งข้อมูลออกเป็น 3 ส่วนในการแสดง คือ ส่วนสัญลักษณ์ (+ or -) ส่วน Exponent และส่วน Significand ในกรณีที่ใช้ 32 บิตในการแสดงเลขทศนิยม 1 ค่า แต่ละส่วนจะมีความยาวดังนี้

Sign 1 bit, Exponent 8 bits, Significand 23 bits

และจะจัดให้อยู่ในรูปแบบดังแสดงในรูป 1 (S ในรูปหมายถึง Sign และเลขในรูปแสดงหมายเลข bit)

รูปที่ 1 General Floating Point Format

- Sign 0 หมายถึงบวก และ 1 หมายถึงลบ

- Exponent เป็นจำนวนเต็ม แสดงในรูป Biased Form (Biased Representation)

Biased Form เป็นวิธีการแสดงเลขจำนวนเต็ม โดยแสดงเลขจำนวนเต็มลบที่ต่ำสุดด้วย 000...000 และจำนวนเต็มบวกสูงสุด ด้วย 111...111 ซึ่งทำได้โดยการบวกเลขจำนวนเต็มลบที่ต่ำสุดเข้าไปในทุกจำนวน เลขจำนวนเต็มลบที่บวกเข้าไปนั้น จะมีชื่อเรียกว่า *Bias* ในมาตรฐาน IEEE754 แบบ 32 บิต Bias มีค่าเท่ากับ 01111111_2 (=127) และกำหนดให้ Exponent 00000000 มีค่าเท่ากับ -126 ยกตัวอย่าง ดังนั้น ถ้าต้องการแสดงเลขชี้กำลัง 1 จะแสดงด้วยเลข $1+127=128=10000000_2=80_{16}$ เลขชี้กำลัง -16 จะถูกแสดงด้วยเลข $-16+127=111=01101111_2=6F_{16}$ เป็นต้น ทั้งนี้ $01111111_2=7F_{16}$ แสดงเลขชี้กำลัง 0

- Significand แสดงส่วนเลขทศนิยม ในมาตรฐาน IEEE754 แบบ 32 บิต ได้กำหนดรูปแบบ *Normalized form* ที่มีรูปแบบดังนี้

$$M = (1.m_1 m_2 \cdots m_{23})_2$$
$$M = 1 + m_1 \times 2^{-1} + m_2 \times 2^{-2} + \cdots + m_{23} \times 2^{-23} \tag{5}$$

และรูปแบบ *Denormalized form* ที่มีรูปแบบดังนี้

$$M = (0.m_1 m_2 \cdots m_{23})_2$$
$$M = m_1 \times 2^{-1} + m_2 \times 2^{-2} + \cdots + m_{23} \times 2^{-23} \tag{6}$$

โดยใช้เลข 23 บิตในการแสดงส่วนเลขทศนิยม อันเป็นสิ่งกำหนดความแม่นยำ (Accuracy) ของค่าที่แสดง ทั้งนี้รูปแบบ Denormalized form ใช้เฉพาะแสดงค่าที่มีเลขชี้กำลังเท่ากับ -126 เท่านั้น

Normalization of Floating Point Number เนื่องจากตัวเลขต่อไปนี้ มีค่าเท่ากัน

0.110×2^5

110×2^2

0.0110×2^6

กล่าวคือ ในการแสดงให้อยู่ในรูปตามสมการ (4) เราสามารถแสดงได้หลายแบบ ซึ่งในการกำหนดมาตรฐาน เราจำเป็นจะต้องมีการแสดงที่เป็นรูปแบบเดียวกัน ประกอบกับจำนวน bit ที่ใช้แสดงตัวเลขมีจำกัด จึงจำเป็นที่จะต้องหาวิธีที่จะใช้จำนวน bit นั้นให้เกิดประโยชน์อย่างสูงสุด ซึ่งในกรณีของ Floating Point Number ก็คือ การที่สามารถใช้จำนวน bit ทั้งหมดของ Significand ให้สามารถแสดงเลขทศนิยมได้ละเอียดเท่าที่จะทำได้ ซึ่งเราเรียกวิธีการจัดรูปแบบเลขทศนิยมดังกล่าวนี้ว่า *Normalization*

ยกตัวอย่างกรณีเลขฐานสิบ ถ้าเราต้องแสดงค่า 3.14 ให้อยู่ในรูป $0.xxx \times 10^{y}$ โดยสามารถแสดงเลขทศนิยมได้เพียง 3 ตำแหน่งเท่านั้น ถ้าเราแสดงด้วย 0.031×10^{2} ความแม่นยำ (Accuracy) ของข้อมูลจะลดลง และทศนิยมตำแหน่งแรกจะไม่ได้ใช้ประโยชน์ ดังนั้น การที่จะใช้ให้เกิดประโยชน์ได้สูงสุด ทำได้โดยการจัดให้อยู่ในรูปแบบที่ทศนิยมตำแหน่งแรกมีค่าที่ไม่เป็น 0 ซึ่งในกรณีนี้ ก็คือ 0.314×10^{1} หลักการของ Normalization of Floating Point Number ก็จะยึดตามหลักการดังกล่าวนี้ คือ การจัดให้อยู่ในรูปแบบหลักแรกของ Significand ไม่เป็น 0 หรือ MSB ของ Mantissa เป็น 1 โดยทำให้อยู่ในรูปต่อไปนี้

$\pm 1.bbbb \ldots b \times 2^{\pm E}$

โดยที่ b เป็น 0 หรือ 1 ซึ่งละไว้ในฐานที่เข้าใจว่า มีเลข 1 นำหน้าส่วนเลขทศนิยม ทำให้เราไม่มีความจำเป็นที่จะต้องบันทึกเลข 1 ดังกล่าวนี้ ดังนั้น เราสามารถใช้จำนวน 23 bit แสดง significand ให้มีความละเอียด 24 bit ได้

<u>ตัวอย่างที่ 9</u> : 0.171875 (กรณีที่ให้จำนวน bit ของ Significand เท่ากับ 8)

$0.171875 = 0.001011_2$

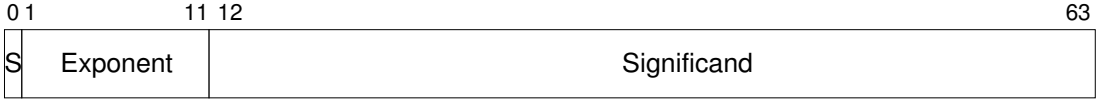
Normalized Form: $0.171875 = 1.011_2 \times 2^{-3}$

<u>หมายเหตุ</u>

1. Normalized form ของ Significand แสดงค่าในช่วง 1.0 ~ 2.0 ($\because 1+\sum_{i=1}^{n} 2^{-i} = 2-2^{-n-1}$)

2. การที่จุดทศนิยมเลื่อนไปตามค่าของ Exponent เป็นที่มาของชื่อเรียก Floating Point Number

2.3.3 **Double Precision Floating Point Number** หรือเรียกสั้น ๆ ว่า **Double** ใช้ในการแสดงเลขทศนิยมเช่นเดียวกับ Floating Point Number เพียงแต่ต่างกันที่จะใช้จำนวน byte เป็น 2 เท่าของ Floating Point Number ธรรมดา ซึ่งมีผลทำให้สามารถแสดงค่าได้มากขึ้นและเที่ยงตรงยิ่งขึ้น จำนวน bit ที่ใช้ในส่วนต่าง ๆ ตามมาตรฐานของ IEEE กำหนดไว้ คือ Sign 1 bit, Exponent 11 bits, Significand 52 bits และจัดอยู่ในรูปแบบดังแสดงในรูป 2



รูป 2 IEEE Double Precision Floating Point Format

IEEE Standard 754

เพื่อให้การแสดงเลขทศนิยมมีมาตรฐานเป็นหนึ่งเดียว IEEE (Institute of Electrical and Electronics Engineers) จึงได้กำหนดมาตรฐานเกี่ยวกับการแสดงขึ้น โดยมีสาระสำคัญ ดังนี้

1. กำหนด Bias ของ Exponent เป็น 127 (Single precision), 1023 (Double precision)

2. กำหนดให้ใช้ค่าตั้งแต่ 1 - 254 (Single Precision) และ 1 - 2046 (Double Format) ในการแสดงค่า Exponent ดังนั้น จะสามารถแสดงค่า Exponent ได้ตั้งแต่ค่า -126 ถึง 127 (Single Precision) และ -1022 ถึง 1023 (Double Precision)

3. Exponent 0 ทั้งหมด กับ Significand 0 ทั้งหมดรวมกัน จะแสดงตัวเลข 0

4. Exponent 1 ทั้งหมด กับ Significand 0 ทั้งหมดรวมกัน จะแสดงค่า Infinity

5. Exponent 0 ทั้งหมด กับ Significand ที่ไม่เป็น 0 ทั้งหมด จะแสดงตัวเลขที่ไม่ได้ทำการ Normalize ในกรณีนี้ Bit ที่อยู่ด้านซ้ายของ Significand จะเป็น 0 และ Exponent เป็น -126 หรือ -1022

6. Exponent 1 ทั้งหมด กับ Significand ที่ไม่เป็น 0 ทั้งหมด จะแสดงค่า NaN (Not a Number)

ซึ่งสามารถสรุปได้ตามตารางที่ 4

ตารางที่ 4 การแปลความหมายของ IEEE 754 Single Precision Floating Point Number (32-bit)

Type	Sign	Biased Exponent	Significand	Value
Positive Zero	0	0	0	0
Negative Zero	1	0	0	-0
Plus Infinity	0	255 (all 1's)	0	∞
Minus Infinity	1	255 (all 1's)	0	$-\infty$
NaN	0 or 1	255 (all 1's)	$\neq 0$	NaN
Positive normalized nonzero	0	$0 < e < 255$	f	$2^{e-127}(1.f)$
Negative normalized nonzero	1	$0 < e < 255$	f	$-2^{e-127}(1.f)$
Positive denormalized	0	0	$f \neq 0$	$2^{-126}(0.f)$
Negative denormalized	1	0	$f \neq 0$	$-2^{-126}(0.f)$

ตัวอย่าง 32-bit Floating Point Number

$1.11010001_2 \times 2^{21}$ = 0 10010100 11010001000000000000000

$-1.11010001_2 \times 2^{21}$ = 1 10010100 11010001000000000000000

$1.11010001_2 \times 2^{-21}$ = 0 01101010 11010001000000000000000

$-1.11010001_2 \times 2^{-21}$ = 1 01101010 11010001000000000000000

ตัวอย่างที่ 10 : 3.625

$3 = (11)_2$, $.625 = .101_2$ $\therefore$ $3.625 = 11.101_2$

Normalized form : $3.625 = 11.101_2 = 1.1101_2 \times 2^1$

เพราะฉะนั้น จะได้ Significand เป็น 1101 และ Exponent เป็น 1 (=1+127=10000000) ดังนั้น จะได้เป็น

0 10000000 $11010000000000000000000_2 = 41680000_{16}$

ข้อจำกัดของ Floating Point Number เช่นเดียวกับเลขจำนวนเต็ม เลขทศนิยมก็มีข้อจำกัดในการแสดงเช่นกัน โดยค่าที่ไม่สามารถแสดงได้แบ่งได้เป็น 4 ส่วน คือ

1. เลขทศนิยมลบที่มีค่าน้อยกว่า $-(2-2^{-24}) \times 2^{127} \sim -10^{38}$ (32-bit), $-(2-2^{-53}) \times 2^{1023} \sim -10^{308}$ (64-bit)
2. เลขทศนิยมลบที่มีค่ามากกว่า -2^{-149} (32-bit), -2^{-1074} (64-bit)
3. เลขทศนิยมบวกที่มีค่าน้อยกว่า 2^{-149} (32-bit), 2^{-1074} (64-bit)
4. เลขทศนิยมบวกที่มีค่ามากกว่า $(2-2^{-24}) \times 2^{127} \sim 10^{38}$ (32-bit), $(2-2^{-53}) \times 2^{1023} \sim 10^{308}$ (64-bit)

เลขทศนิยมที่ไม่สามารถแสดงได้นี้ จำแนกเป็น 2 ประเภท คือ

1. *Overflow* คือ เลขทศนิยมที่มีค่า absolute มากเกินกว่าช่วงที่สามารถแสดงได้ คือ ส่วนที่ 1 (Negative Overflow) และ 5 (Positive Overflow)

2. *Underflow* คือ คือ เลขทศนิยมที่มีค่า absolute น้อยเกินกว่าช่วงที่สามารถแสดงได้ คือ ส่วนที่ 2 (Negative Underflow) และ 4 (Positive Underflow)

ในกรณีที่มีการคำนวณแล้วเกิดค่าที่เกิน หรือต่ำกว่าช่วงที่สามารถแสดงได้ เช่น $0.8\times2^{100}\times2^{50}$ จะได้ผลลัพธ์ที่ไม่สามารถแสดงค่าได้ กรณีที่เรียกว่า *Overflow Error* ส่วนในกรณีที่ได้ผลลัพธ์เป็นค่าที่เล็กมากจนไม่สามารถแสดงได้จะเรียกว่า *Underflow Error*

<u>หมายเหตุ</u> โดยสรุปแล้ว ข้อผิดพลาดในการคำนวณ Floating Point Number จะมีอยู่ 3 ประการใหญ่ ๆ คือ Round-Off Error, Overflow Error และ Underflow Error

แบบฝึกหัด

1. จงแปลงเลขฐานสิบต่อไปนี้ ให้เป็นเลขฐานสอง โดยแสดงให้อยู่ในรูปเลขฐานสิบหก

1.1	57	1.2	357	1.3	4000
1.4	0.750	1.5	0.40625	1.6	0.85

2. จงแปลงเลขฐานสิบต่อไปนี้ ให้อยู่ในรูปแบบเลขจำนวนเต็มที่มีขนาด 2 Bytes

2.1	108	2.2	30000	2.3	-50

3. จงแปลงเลขจำนวนเต็มที่มีขนาด 2 Bytes แบบ Signed Integer ต่อไปนี้ ให้เป็นเลขฐานสิบ

3.1	0x00FF	3.2	0xFFFD	3.3	0x0F0F
3.4	0x00FF + 0x0F00	3.5	0xF000 + 0xFF00	3.6	0x7F0F - 0x5D5D

4. จงแปลงเลขทศนิยมต่อไปนี้ ให้อยู่ในรูป IEEE Floating Point Format (ขนาด 4 Bytes)

4.1	3.5	4.2	-100.5625	4.3	8.35

5. จงแปลงเลขทศนิยมที่อยู่ในรูป IEEE Floating Point Format ต่อไปนี้ ให้เป็นเลขฐานสิบ

5.1	0x4248C000	5.2	0xC0600000

6. ถ้าหากเลขจำนวนเต็มขนาด 2 Bytes และเลขทศนิยมขนาด 4 Bytes ข้อใดที่จะเกิดปัญหาขณะคำนวณ

6.1	4000×20	6.2	$4000 \div (-50)$	6.3	$35000 + 4000$
6.4	$1 / 10^{-50}$	6.5	$10^{30} / 10^{70}$	6.6	9.2×0.2