ตัวเลข

คพ 620 สถาปัตยกรรมคอมพิวเตอร์และซอฟต์แวร์ระบบ

กษิดิศ ชาญเชี่ยว

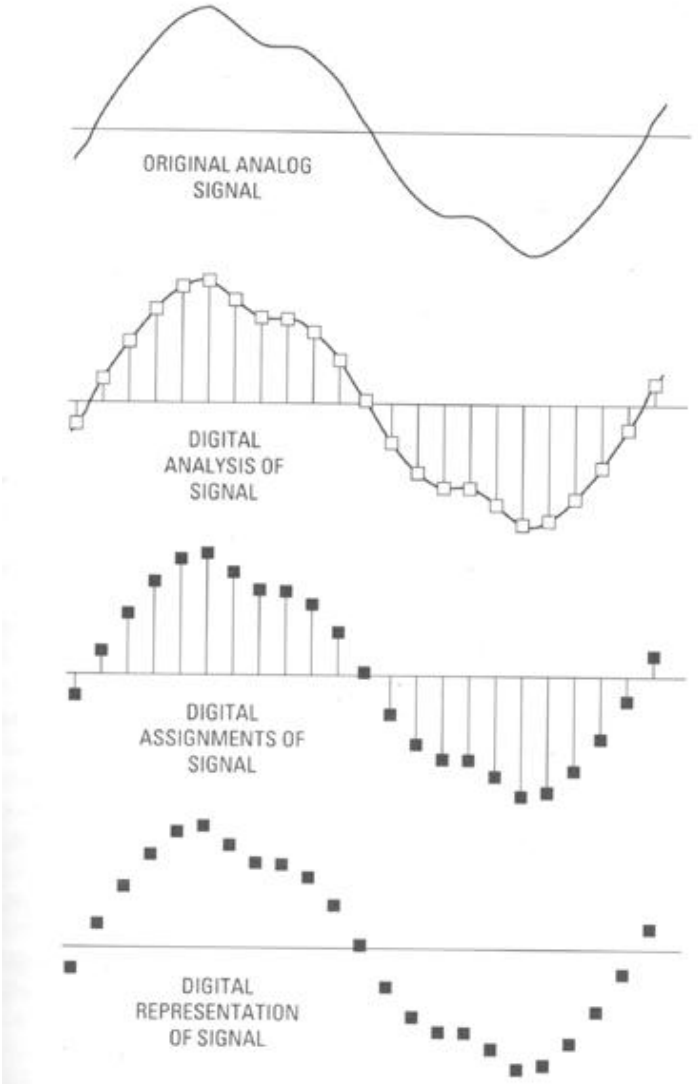
ภาค 2/2558

การแทนค่าตัวเลข

- ข้อมูลส่วนใหญ่ในธรรมชาติเป็น Analog มีความต่อเนื่อง
- ข้อมูลที่เก็บในระบบคอมพิวเตอร์เป็น Digital
- แปลง Alnalog เป็น Digital เพื่อป้อนเป็นข้อมูลเข้าให้ระบบคอมพิวเตอร์
  - การ Sample

    ตัวอย่าง: ใน CD ความดังของดนตรีจะถูกวัดและเก็บค่า Sample ทุกๆ 1/44100 วินาที
  - การกำหนดปริมาณ Quantize

    ค่าที่ได้จากการแซมเปิลจะถูกนำมาแปลงเป็นตัวเลขตามมาตรวัดแบบใดแบบหนึ่ง เช่นในภาพ ค่าจะถูกแปลงเป็นตัวเลขสิบหกบิต (ซึ่งมีจำนวนค่า 2^{16} =65536 ค่า)



www.joshuadysart.com/journal/archives/digital_sampling.gif

ข้อมูลบางอย่างเป็น Digital แต่แรก

Taken from povray.org

ข้อมูล Bit สามารถเป็นตัวแทนอะไรก็ได้

- ข้อมูลตัวอักษร (characters)
  - ตัวอักษร 26 ตัว ไช้ 5 bits แทนได้
  - แทนตัวอักษร ตามมารตรฐาน ASCII
  - แทนตัวอักษรแบบ unicode ด้วยรหัส 8, 16, 32 bits
- ข้อมูลตรรกะ จริง = 1 เท็จ = 0
- ข้อมูลสี แดง = 00 เขียว = 01 น้าเงิน = 11
- ข้อมูลอื่นๆเช่น ตำแหน่งที่อยู่ คำสั่ง เป็นต้น
- คำถาม ต้องใช้กี่บิตในการแทนค่า π

ปฏิบัติการของ ตัวเลข

- ปฏิบัติการ
  - บวก
  - ลบ
  - คูณ
  - หาร
- ตัวอย่าง 10 + 7
- การบวกเป็นวงจรอย่างง่าย
- การเปรียบเทียบตัวเลข ทำอย่างไร X > Y

```
      1   1
      1   0   1   0
+     0   1   1   1
-------------------
1     0   0   0   1
```

การแทนค่าตัวเลขจำนวนเต็มไม่มีเครื่องหมาย

- เราใช้ชุดของบิตแทนค่าตัวเลข
  - รูปแบบของชุดของบิต 1 และ 0 ที่แตกต่างกันแทนค่าตัวเลขจำนวนเต็มที่แตกต่างกัน เป็นความสัมพันธ์แบบ 1 ต่อ 1
- เวลาที่เพิ่มค่า ค่าบิตทางซ้ายมือจะเปลี่ยนไปเรื่อยๆในขณะที่ทางขวาจะเปลี่ยนไปในอัตราที่ช้ากว่า
- จำนวนดิจิต(digit)คือ จำนวนบิตที่ระบบคอมพิวเตอร์ใช้เป็นตัวแทนของค่าตัวเลขจำนวนเต็ม
- หากค่าของจำนวนเต็มมากกว่าที่จำนวนดิจิต(ที่ใช้สำหรับการแทนค่า) สถานะการณ์นั้นเรียกว่า overflow

overflow

- 00000 00001 00010 ..... 11110 11111

  จำนวนเต็มไม่มีเครื่องหมาย unsigned integer

- 00000 00001 00010 ..... 11110 11111

- ภาษา C ใช้ unsigned int หรือ C99 ใช้ uint64_t
- เมื่อมีการบวกค่าบิตดิจิต ค่าจำนวนเต็มเพิ่มขึ้นทีละ 1 ด้วย
  - การเปลี่ยนแปลงของค่าบิตดิจิตและค่าจำนวนเต็มเป็นไปในทิศทางที่เพิ่มขึ้นเหมือนกัน

การแทนค่าจำนวนเต็มแบบมีเครื่องหมายและค่า (signed and magnitude)

- กำหนดให้บิตซ้ายสุด (leftmost) เป็นตัวกำหนดเครื่องหมาย
  - 0 คือค่าบวก และ 1 คือค่าลบ
  - ดิจิตที่เหลือเป็นค่าจำนวนเต็ม

บิตดิจิตเพิ่มค่าทีละ 1

- 00000 00001 ..... 01110 01111

11111 11110 ... 10001 10000

รูปแบบบิตดิจิตเพิ่มตามหลักการบวกเลขฐาน 2 แต่ค่าของตัวเลขที่บิตดิจิตเป็นตัวแทนนั้นลดลง

รูปแบบบิตดิจิตเพิ่มทีละ 1 ตามหลักการบวกเลขฐาน 2 และค่าของตัวเลขที่บิตดิจิตเป็นตัวแทนเพิ่มสอดคล้องกัน

ข้อเสียของ signed and magnitude

- ออกแบบวงจร บวก ลบ คูณ หาร ยุ่งยาก
  - ต้องเช็คเครื่องหมายก่อน (เช่นการบวกค่า ถ้าเป็นบวก ก็ใช้การบวกเลขฐานสองแต่ถ้าเป็นลบก็ต้องใช้การลบเลขฐานสองแทน)
- มีบิตดิจิตสองรูปแบบเป็นตัวแทนค่าศูนย์
  - 0x00000000 = +0 (ฐานสิบ)
  - 0x80000000 = -0 (ฐานสิบ)
- เมื่อค่าบิตดิจิตเพิ่มขึ้น บางครั้งค่าเพิ่ม บางครั้งค่าลด
- เราจึงไม่ใช้วิธีนี้แทนค่าจำนวนเต็มมีเครื่องหมาย

การแทนค่าแบบ one's complement

- ยกตัวอย่างเช่น: 7_{10}= 00111_2 -7_{10}= 11000_2

บิตดิจิตเพิ่มค่าทีละ 1

- 00000 00001 ..... 01110 01111

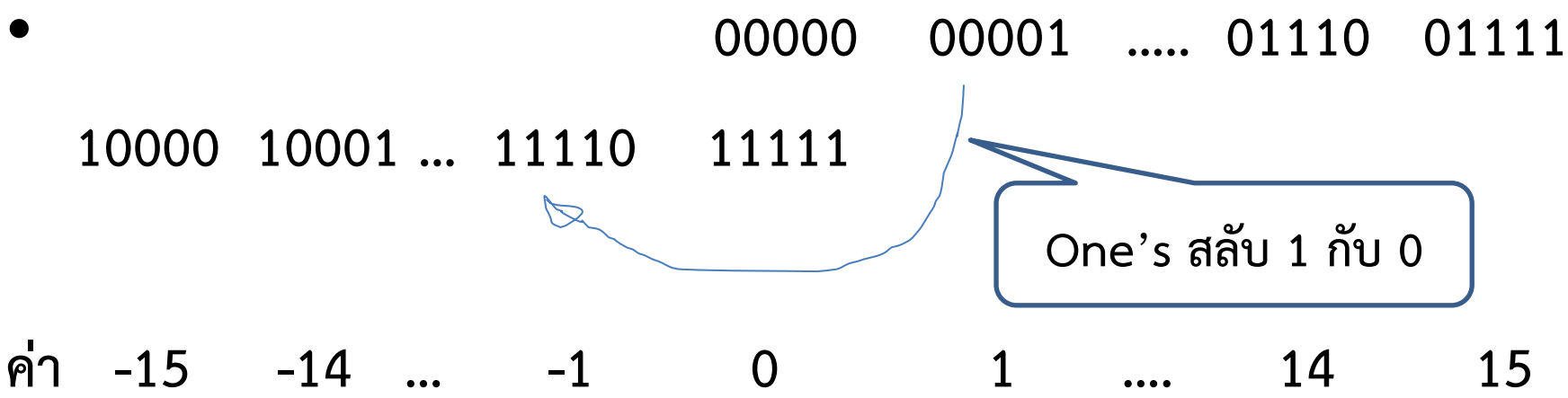
10000 10001 ... 11110 11111

- ทั้งค่าบวกและลบเพิ่มค่าในทิศทางดียวกัน เมื่อเพิ่มค่าบิตดิจิต
- ปัญหา: 0 มีตัวแทนสองตัวคือ 00000 และ 11111
- แทนตัวเลขบวกและค่าลบได้กี่ค่าถ้ามี N บิต

ข้อเสียของ one’s complement

- ออกแบบวงจรยาก
- มีตัวแทนค่า 0 สองตัวแทนคือ
  - 0x00000000 = +0 (ฐานสิบ)
  - 0xFFFFFFFF = -0 (ฐานสิบ)
- One’s complement ได้ถูกไช้บนเครื่องคอมพิวเตอร์บางระบบ แต่ต่อมาถูกยกเลิกเพราะมีระบบแทนค่าตัวเลขจำนวนเต็มที่ดีกว่าเกิดขึ้น

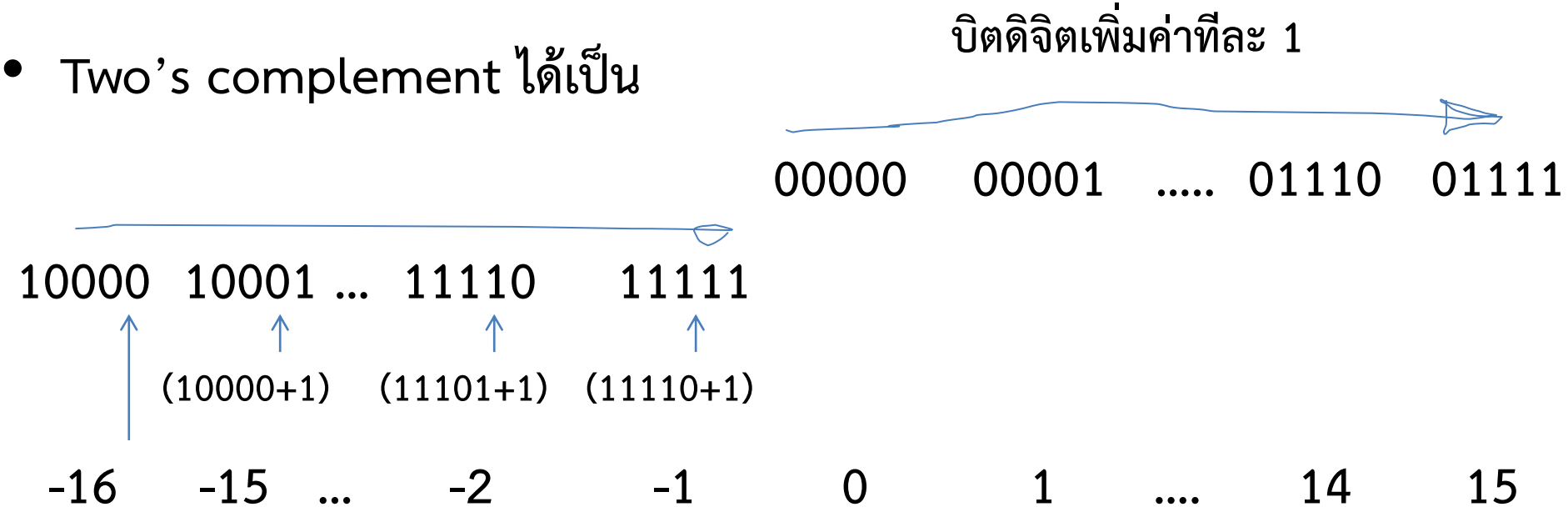
การแทนค่าแบบ Two's complement

- จาก one's complement ข้างล่าง
- 

- หากเราอยากจะให้ 11111 มาเป็นตัวแทนของ -1 ล่ะจะทำอย่างไรดี
  - One's complement ให้ค่า -1 คือ 11110 หากจะแปลงให้เป็น 11111 ก็คือต้องเอา 11110 มาบวก 1
  - และต้องใช้วิธีการนี้กับค่าลบอื่นๆที่ได้จาก one's complement ด้วย

การแทนค่าแบบ Two's complement

- Two's complement ได้เป็น



- -15 กลายเป็น 10001 ทำให้เหลือ 10000 ซึ่งกำหนดให้เป็นตัวแทนของ -16
- Two's complement = one's complement แล้วบวก 1
- ออกแบบวงจรง่ายและแก้ปัญหาการแทนค่า 0

การแทนค่าแบบ Two's complement

- ออกแบบวงจรง่ายและแก้ปัญหาการแทนค่า 0
- ชนิดในภาษา C ได้แก่ int หรือ int32_t, int64_t
- สำหรับค่าตัวเลขฐานสองแบบ 32 บิต $d_{31}\ d_{30}\ d_{29}\ d_{28}\ldots\ d_2\ d_1\ d_0$

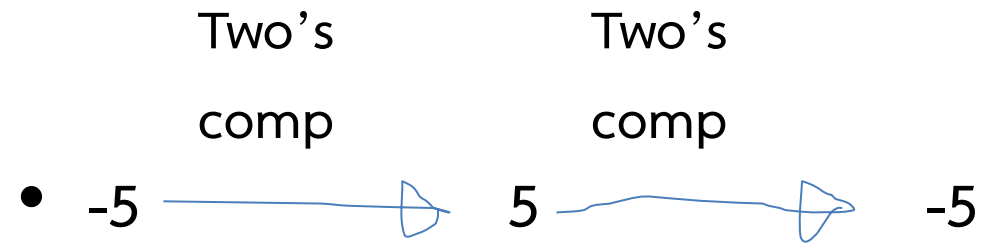
$d_{31} \times (-1)(2^{31}) + d_{30} \times 2^{30} + d_{29} \times 2^{29} + \ldots d_2 \times 2^2 + d_1 \times 2^1 + d_0 \times 2^0$

- ให้หาค่า 1011_2 ว่าเท่ากับเลขฐานสิบเท่าไร

$= 1 \times (-1)(2^3) + 0 \times (2^2) + 1 \times (2^1) + 1 \times (2^0)$

$= \quad -8 \quad + \quad 0 \quad + \quad 2 \quad + \quad 1 \quad = \quad -5$

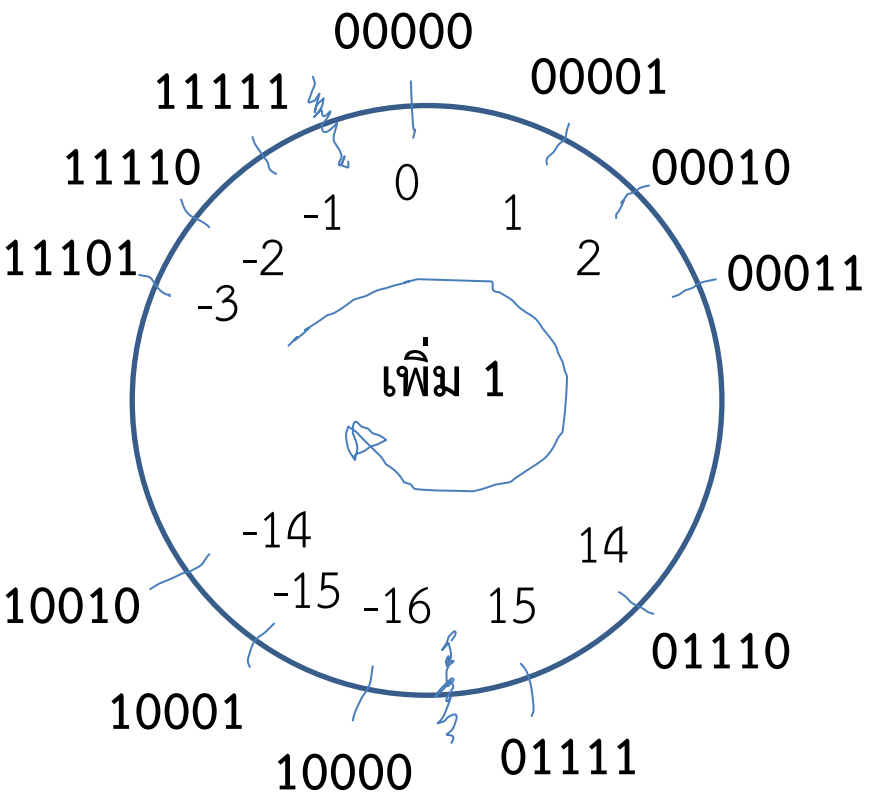
การแทนค่าแบบ Two's complement



- แสดงว่า ค่าเปลี่ยนอย่างไร
- 1011 (0100+1) 0101 (1010+1) 1011

Two's complement N=5

ค่า Number = Bit Representation

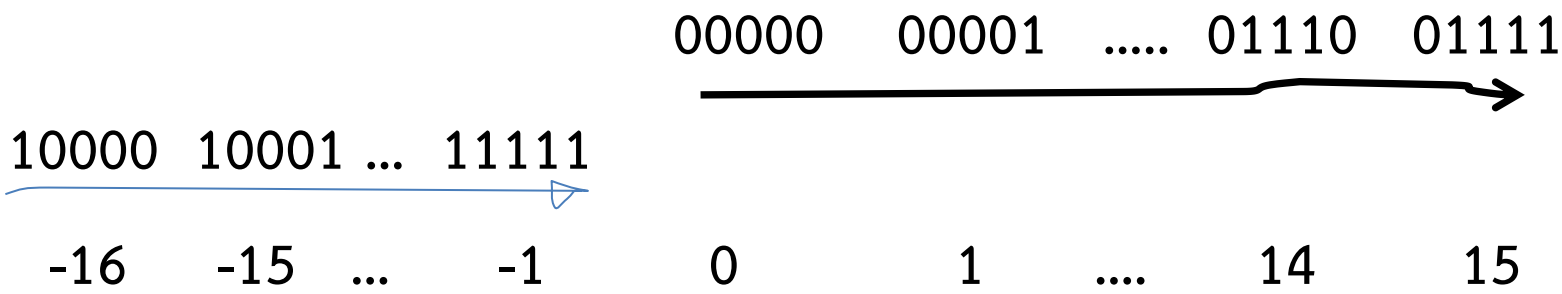


2^{N-1} Non-negative

2^{N-1} Negative

มีกี่ Positive

การแทนค่าแบบ Excess K หรือ Bias -K

- ข้อด้อยของ 2's complement คือ การเปรียบเทียบค่าสองค่าแบบมีเครื่องหมาย ไม่สามารถใช้วงจรฮาร์ดแวร์เดียวกันเพื่อ*เปรียบเทียบค่า*หรือ *sort เรียงลำดับค่า*ได้
- ยกตัวอย่างเช่นในการแทนค่าแบบ 2's complement ที่จำนวนบิต N = 5 bits ข้างล่าง
- 

- ถ้าเปรียบเทียบ 11111 กับ 00001 แบบ unsigned จะได้ว่า 11111 > 00001 แต่ถ้าเทียบแบบ signed จะได้ว่า 00001 > 11111

การแทนค่าแบบ Excess K หรือ Bias -K

- ด้วยเหตุนี้จึงได้มีการคิดแทนค่าแบบ Excess K หรือ Bias –K ขึ้นมาโดยกำหนดให้ใช้ค่าที่มีบิต 0 จำนวน N bits แทนค่าที่น้อยที่สุดคือ –K และค่าบิตถัดไปคือ 0000... 1 แทนค่า –K + 1 ไปเรื่อยๆจนถึงค่าบิตที่มี 111...1 จำนวน N บิตทั้งหมด
- ยกตัวอย่างเช่นในการแทนค่าแบบ ที่จำนวนบิต N = 5 bits และให้ Bias = -K = -15 ข้างล่าง (N = 5, K = 15)
- 00000 00001 ..... 01110 01111 10000 10001 ... 11111

ค่า -15 -14 ... -1 0 1 2 16

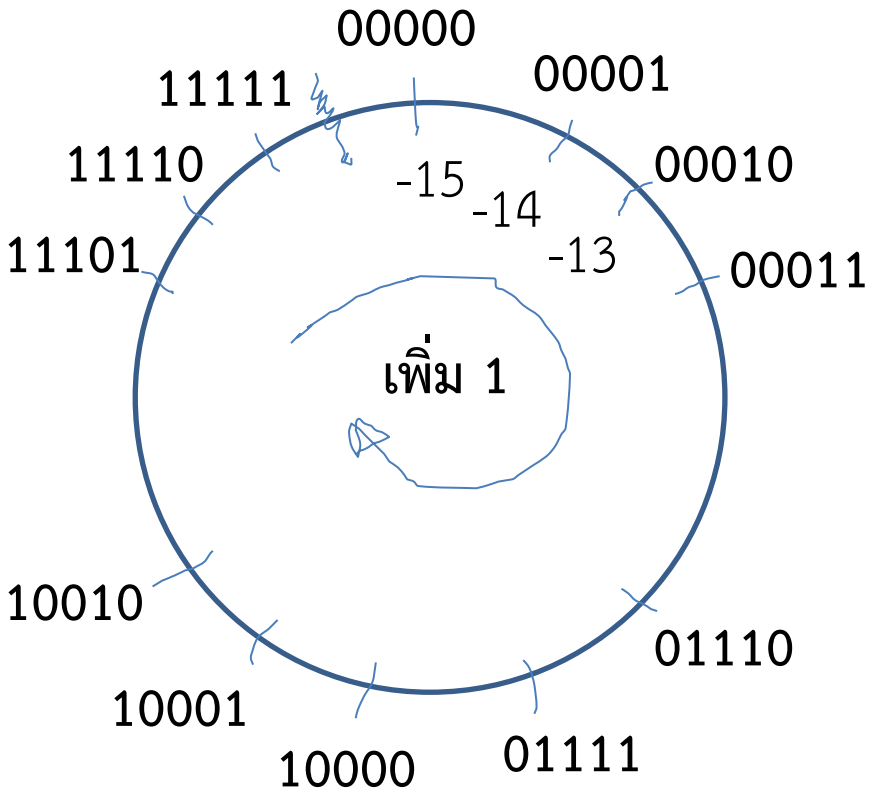
- ถ้าเปรียบเทียบ 11111 กับ 00001 แบบ unsigned จะได้ว่า 11111 > 00001 ซึ่งก็สอดคล้องกับค่า 16 > -14

การแทนค่าแบบ Excess K หรือ Bias -K

- เพราะฉะนั้นการเปรียบเทียบค่าแบบมี และไม่มีเครื่องหมายจึงใช้วงจรฮาร์ดแวร์วงจรเดียวกันได้
- แต่อย่างไรก็ตามข้อเสียของ Excess K หรือ Bias –K คือ การออกแบบฮาร์ดแวร์เพื่อบวกค่ายุ่งยากเพราะจะต้องมีการกำหนดค่า N และ K (ในขณะที่ 2'sComplement ใช้ N อย่างเดียว)
- ผู้ออกแบบระบบคอมพิวเตอร์ส่วนใหญ่คิดว่า วงจรการบวก สำคัญกว่าวงจรการเปรียบเทียบ ดังนั้นจึงเลือกใช้ 2's complement ในการแทนค่าตัวเลข
- แต่อย่างไรก็ตาม Bias K มีประโยชน์ในเรื่อง Floating Point ที่เราจะได้ศึกษาต่อไป
- ถัดไปเราจะดูตัวอย่างการ Bias หลายๆอย่าง

Excess K = 15 เบี่ยง Bias = -15

ค่า Number = ค่า Bit Representation - 15



ค่า Number ของ “00000”

= (ค่าฐานสิบของ 00000) – 15

= 0 – 15

= -15

ถัดไป 00001 – 15 = -14

ถัดไป 00010 – 15 = -13

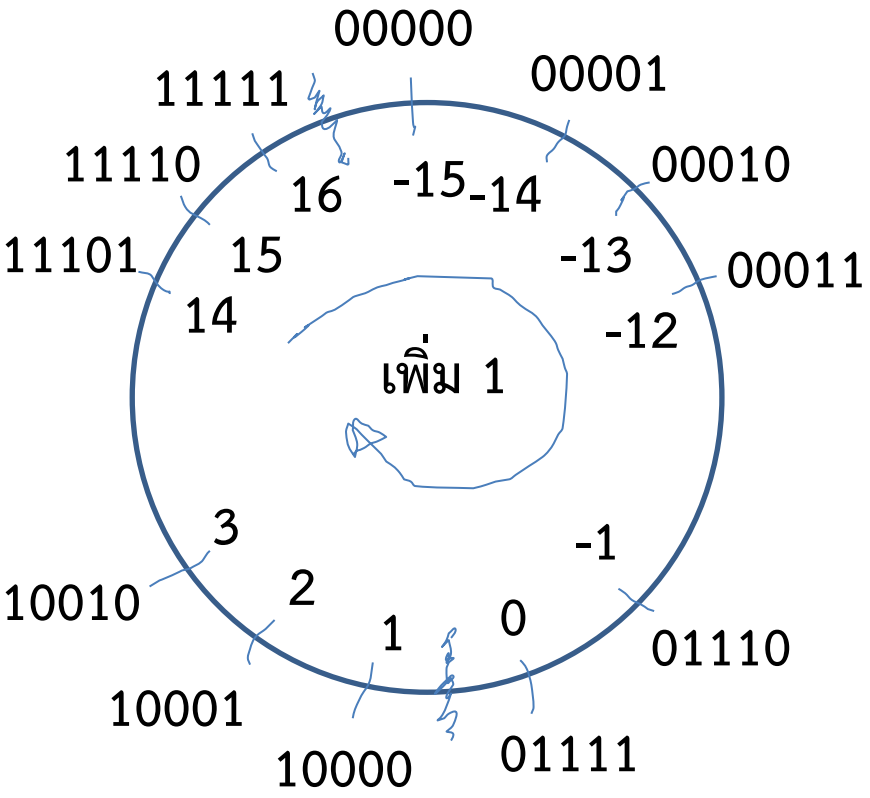
.....

ถัดไป 01111 – 15 = 0

ให้สังเกตุว่าในกรณี K = 15 นี้ ค่า 0 ทำให้มีการเปลี่ยน signed bit ด้วย

N=5 (เบี่ยง Bias = -15)

ค่า Number = Bit Representation - 15



มีกี่ Positive

= 2^{N-1}

2^{N-1} Non-positive

(รวม one zero)

ตัวอย่าง N และ K

- (N = 5, K = 10) Bias -10

	00000	00001	.....	01110	01111	10000	10001	...	11111
ค่า	-10	-9	...	4	5	6	7	....	21
	0 + (–10)	1 + (–10)		14 + (–10)	15 + (–10)	16 + (–10)	17 + (–10)		31 + (–10)

- (N = 5, K = -22) Bias –(-22) คือ Bias +22

	00000	00001	.....	01110	01111	10000	10001	...	11111
ค่า	22	23	...	36	37	38	39	....	43
	0 + (22)	1 + (22)		14 + (22)	15 + (22)	16 + (22)	16 + (22)		31 + (22)

ตัวอย่าง N และ K

- (N = 3, K = 4) Bias -10

	000	001	010	011	100	101	110	111
ค่า	-4	-3	-2	-1	0	1	2	3
	0 + (–4)	1 + (–4)	2 + (–4)	3 + (–4)	4 + (–4)	5 + (–4)	6 + (–4)	7 + (–4)

Number of Bits	Number of Values	Min Value	Max Value
16	2^{16}	-K	$-K + (2^{16} - 1)$
32	2^{32}	-K	$-K + (2^{32} - 1)$
N	2^{N}	-K	$-K + (2^{N} - 1)$

http://www.cs.umd.edu/class/sum2003/cmsc311/Notes/Data/bias.html

การแทนค่าเลขทวินิยม (ทศนิยม)

- เราจะแทนค่า -2.5 หรือ -2.75 หรือ -12.75 อย่างไรดี
- ใช้หลักการเดียวกันกับเลขฐานสิบ (เลื่อนค่าจุดทศนิยม)
  - ทศนิยมกี่ตำแหน่ง

$12.5_{10} = (1 \times 10^{1})+(2 \times 10^{0}) + (5 \times 10^{-1})$

- 010.1_2 เท่ากับเท่าไร (ให้บิตซ้ายสุดเป็น sign bit)

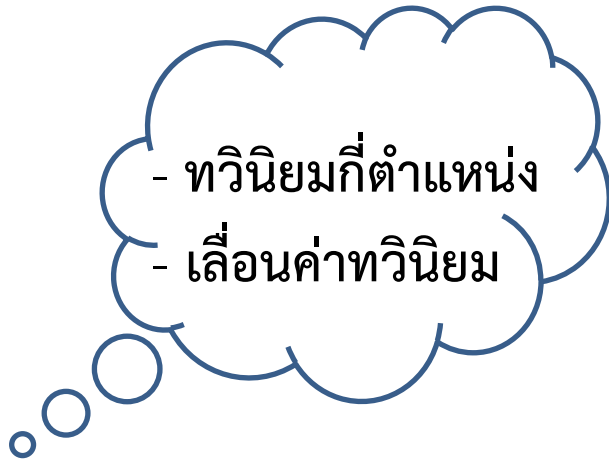
$010.1_2 = (0 \times (-2^2)) + (1 \times 2^1) + (0 \times 2^0) + (1 \times 2^{-1})$

$= 0 + 2 + 0 + (1/2) = 2.5$

$010.11_2 = (0 \times (-2^2)) + (1 \times 2^1) + (0 \times 2^0) + (1 \times 2^{-1}) + (1 \times 2^{-2}) = 2.75$

- ใช้หลักการ 2's complement ได้ e.g. ค่าลบของ $010.11_2 = 101.01_2$

$101.01_2 = (1 \times (-1)(2^2)) + (0 \times 2^1) + (1 \times 2^0) + (0 \times 2^{-1}) + (1 \times 2^{-2}) = -2.75$



Bit Pattern				Number Represented (n)	n / 2
1	1	1	1	-1	-0.5
1	1	1	0	-2	-1
1	1	0	1	-3	-1.5
1	1	0	0	-4	-2
1	0	1	1	-5	-2.5
1	0	1	0	-6	-3
1	0	0	1	-7	-3.5
1	0	0	0	-8	-4
0	1	1	1	7	3.5
0	1	1	0	6	3
0	1	0	1	5	2.5
0	1	0	0	4	2
0	0	1	1	3	1.5
0	0	1	0	2	1
0	0	0	1	1	0.5
0	0	0	0	0	0

การแทนค่าแบบ Two’s complement ของเลข 32 บิต

$0000\ \ldots\ 0000\ 0000\ 0000\ 0000_{two}$ =	0_{ten}
$0000\ \ldots\ 0000\ 0000\ 0000\ 0001_{two}$ =	1_{ten}
$0000\ \ldots\ 0000\ 0000\ 0000\ 0010_{two}$ =	2_{ten}
. . .	
$0111\ \ldots\ 1111\ 1111\ 1111\ 1101_{two}$ =	$2{,}147{,}483{,}645_{ten}$
$0111\ \ldots\ 1111\ 1111\ 1111\ 1110_{two}$ =	$2{,}147{,}483{,}646_{ten}$
$0111\ \ldots\ 1111\ 1111\ 1111\ 1111_{two}$ =	$2{,}147{,}483{,}647_{ten}$
$1000\ \ldots\ 0000\ 0000\ 0000\ 0000_{two}$ =	$-2{,}147{,}483{,}648_{ten}$
$1000\ \ldots\ 0000\ 0000\ 0000\ 0001_{two}$ =	$-2{,}147{,}483{,}647_{ten}$
$1000\ \ldots\ 0000\ 0000\ 0000\ 0010_{two}$ =	$-2{,}147{,}483{,}646_{ten}$
. . .	
$1111\ \ldots\ 1111\ 1111\ 1111\ 1101_{two}$ =	-3_{ten}
$1111\ \ldots\ 1111\ 1111\ 1111\ 1110_{two}$ =	-2_{ten}
$1111\ \ldots\ 1111\ 1111\ 1111\ 1111_{two}$ =	-1_{ten}

การขยายค่าแบบมีเครื่องหมายเมื่อจำนวนบิตมากขึ้น (Signed Extension)

- เช่นขยายแบบมีเครื่องหมาย (Signed Extend) จากค่า 16 บิตเป็น 32 บิต
- ให้ copy ค่า sign bit ไปบนบิตที่สร้างใหม่ทางซ้ายทั้งหมด
  - หากเป็นเลขบวก ให้เติมค่า 0 ไปบนบิตทางซ้ายที่ขยายใหม่ทั้งหมด
  - หากเป็นเลขลบ ให้เติมค่า 1 ไปบนบิตทางซ้ายที่ขยายใหม่ทั้งหมด
- ตัวอย่าง ขยายจาก 16 bits เป็น 32 bits ของค่าข้างล่าง

$$1111\ 1111\ 1111\ 1100_{two}$$

$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_{two}$$

สรุป

- แทนค่าจำนวนเต็มแบบ signed and magnitude
- การแทนแบบ One’s complement
- การแทนแบบ Two’s complement