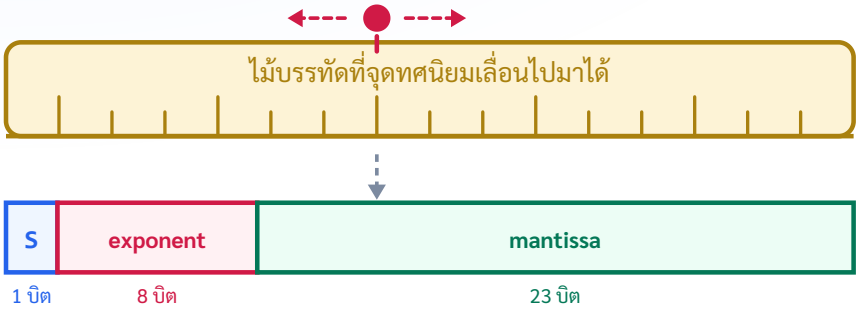
# 03 EP

# Floating point
# กับเลขคณิตจำนวนมีเครื่องหมาย

IEEE 754 single precision 32 บิต · normalize · hidden bit · bias 127 · บวกลบคูณหารในรูป 2's complement · overflow condition




1.m × 2ᵉ — รวมกันได้ 32 ช่องพอดีครับ

พี่สอนน้องแบบไม่ต้องมีพื้นฐานมาก่อนเลยครับ · อ้างอิงสไลด์ EES271 Lecture 02 หน้า 1–13 + Assignment #2 ข้อ 1–3

# ตอนนี้เราจะเดินไปทางไหนกัน

ก่อนเริ่ม พี่ขอกางแผนที่ให้น้องดูก่อนครับ ว่า EP นี้มีอะไรบ้าง จะได้ไม่หลงทาง


| | | |
|---|---|---|
| 1 | ทำไมจุดทศนิยมถึงต้อง “ลอย” — และกล่อง 32 ช่องหน้าตายังไง | พื้นฐานสุด |
| 2 | สามความลับ: normalize · hidden bit · bias 127 | ต้องเข้าใจที่มา |
| 3 | สูตร 6 ขั้น: ฐานสิบ → single precision | ใช้ทุกข้อ |
| 4 | ซ้อมมือ floating point + วิธีอ่านย้อนกลับ | ซ้อมมือ |
| 5 | บวกเลขมีเครื่องหมาย — 4 กรณี | ออกสอบชัวร์ |
| 6 | Overflow — เมื่อคำตอบใหญ่เกินกล่อง | จุดตาย |
| 7 | ลบ = บวกกับ 2's complement ของตัวลบ | ออกสอบชัวร์ |
| 8 | คูณและหารเลขมีเครื่องหมาย | ต้อง true form |
| 9 | เฉลย Assignment 2 ข้อ 1 — floating point | เฉลยเต็ม |
| 10 | เฉลย Assignment 2 ข้อ 2 (a–b) — บวก | เฉลยเต็ม |
| 11 | เฉลย Assignment 2 ข้อ 3 (a–b) — ลบ | เฉลยเต็ม |
| 12 | กับดักรวม 8 ข้อ + สรุปหน้าเดียว + เช็คลิสต์ก่อนสอบ | ทวนก่อนสอบ |


## ชีทนี้ใช้ยังไงให้ได้ผลที่สุด

ในชีทจะมีป้าย 3 แบบสลับกันไปครับ อยากให้น้องใช้มันคนละแบบ

**I DO**

**พี่ทำให้ดู** — น้องอ่านอย่างเดียวพอครับ แต่ขอให้อ่าน**เหตุผล**ในแต่ละขั้นด้วย ไม่ใช่แค่ดูว่าทำอะไร

**WE DO**

**ทำด้วยกัน** — มีช่องว่างให้เติม ลองเติมในใจก่อนแล้วค่อยดูเฉลยที่อยู่ถัดลงไปทันทีครับ

**YOU DO**

**ตาน้อง** — เฉลยอยู่ติดกันเลย แต่พี่ขอให้**ปิดไว้ก่อน** ทำเองให้จบแล้วค่อยเปิดเทียบครับ

เรื่องเครื่องหมายในชีทนี้

EP นี้มีเลข 0 กับ 1 เต็มไปหมดครับ พี่เลยใช้สีช่วยแยกให้: **ฟ้า = บิตเครื่องหมาย (sign)** · **ชมพู = เลขชี้กำลัง (exponent)** · **เขียว = ตัวเลขนัยสำคัญ (mantissa)**

ส่วนในกล่องเลขคณิตตั้งหลัก **ตัวเลขสีแดงตัวเล็กด้านบน = ตัวทด** และช่องที่ระบายเหลืองคือ**บิตเครื่องหมาย** ซึ่งเป็นบิตที่ใช้ตัดสิน overflow ครับ

# 1 ทำไมจุดทศนิยมถึงต้อง “ลอย”

**ตะขอเกี่ยว**

ตอนที่แล้ว (EP02) น้องบวกลบเลขจำนวนเต็มในรูป 2's complement ได้แล้วใช่ไหมครับ $-45$ เก็บเป็น `11010011` บวกลบได้สบาย ๆ

งั้นสมมติว่าถ้าน้องเจอเลขอย่าง $-99.03125$ ละครับ **คอมพิวเตอร์จะเก็บยังไง ในเมื่อจุดทศนิยมมันเลื่อนไปมาได้** กล่อง 8 ช่องของ EP02 ไม่มีช่องไหนไว้ใส่ “จุด” เลยสักช่อง แล้วถ้าวันนี้เก็บ $99.03125$ พรุ่งนี้เก็บ $0.000012$ จุดมันอยู่คนละที่กันเลย — EP นี้จะตอบคำถามนี้ครับ

## ของใกล้ตัว: ไม้บรรทัดที่ขยับตำแหน่งจุดได้

ลองนึกถึงไม้บรรทัดในกล่องดินสอน้องครับ ถ้าพี่บังคับให้น้องเขียนตัวเลขทุกตัวโดย **ตรึงจุดไว้ตรงกลางไม้บรรทัดตายตัว** (เรียกว่า fixed point) น้องจะเจอปัญหาทันที เพราะเลขใหญ่มาก ๆ กับเลขเล็กมาก ๆ ต้องใช้ช่องคนละแบบกันเลยครับ

**fixed point — จุดตรึงตาย**

แบ่งกล่อง 32 ช่องเป็น “16 ช่องหน้าจุด · 16 ช่องหลังจุด” ตายตัว → เก็บเลขได้แค่ประมาณ $\pm 65535$ และละเอียดแค่ $2^{-16}$ อยากเก็บเลขใหญ่กว่านี้ก็จบเลยครับ

**floating point — จุดลอยได้**

ไม่เก็บ “ตำแหน่งจุด” แต่เก็บ **ตัวเลข** กับ **คำสั่งว่าให้เลื่อนจุดไปกี่ช่อง** แยกกัน → กล่องเท่าเดิม 32 ช่อง แต่เก็บได้ตั้งแต่ $10^{-38}$ ถึง $10^{38}$ ครับ

## พี่ขอเฉลยเลยว่ามันคือของที่น้องใช้เป็นอยู่แล้ว

น้องเคยเขียน $12300 = 1.23 \times 10^4$ ในวิชาฟิสิกส์ใช่ไหมครับ นั่นแหละครับ **floating point คือของอันเดียวกันเป๊ะ** ต่างกันแค่ 2 อย่างเท่านั้น: เปลี่ยนฐานจาก 10 เป็น 2 แล้วยัดทั้งสามส่วน (เครื่องหมาย · เลขชี้กำลัง · ตัวเลขนัยสำคัญ) ลงกล่อง 32 ช่องครับ

**ที่น้องเคยใช้ในวิชาฟิสิกส์**

$$12300 = 1.23 \times 10^4$$

| | |
|---|---|
| ตัวหน้าจุด | 1 หลักเสมอ |
| ตัวหลังจุด | 23 |
| ฐาน | 10 |
| เลขชี้กำลัง | 4 |

**ที่คอมพิวเตอร์ใช้ — ของอันเดียวกัน**

$$1101_2 = 1.101 \times 2^3$$

| | |
|---|---|
| ตัวหน้าจุด | 1 เสมอ (ไม่ต้องเก็บ) |
| ตัวหลังจุด | mantissa |
| ฐาน | 2 |
| เลขชี้กำลัง | 3 (แล้ว +127) |

floating point ไม่ใช่ของใหม่ครับ — คือสัญกรณ์วิทยาศาสตร์ที่น้องใช้อยู่แล้ว แค่เปลี่ยนฐาน 10 เป็นฐาน 2 แล้วยัดลงกล่อง 32 ช่อง

**เข้าเรื่อง**

**Floating-point numbers are a way to represent real numbers (numbers that have fractional parts) in computers. Stored as single-precision (4-byte) or double-precision (8-byte) form, using the IEEE 754 standard.**

แปลเป็นภาษาพี่: เวลาเจอเลขที่มีเศษหลังจุด คอมพิวเตอร์จะไม่เก็บตรง ๆ ครับ แต่จะแตกมันเป็น **3 ชิ้น** แล้วเก็บทีละชิ้นลงกล่องที่มีขนาดตายตัว

มาตรฐานที่ทุกเครื่องบนโลกใช้ตรงกันชื่อ `IEEE 754` อ่านว่า “ไอ-ทริปเปิลอี เจ็ดห้าสี่” ครับ ที่ต้องมีมาตรฐานเพราะว่า ถ้าเครื่องหนึ่งเก็บแบบหนึ่ง อีกเครื่องอ่านอีกแบบ ไฟล์ส่งข้ามเครื่องก็เละหมด

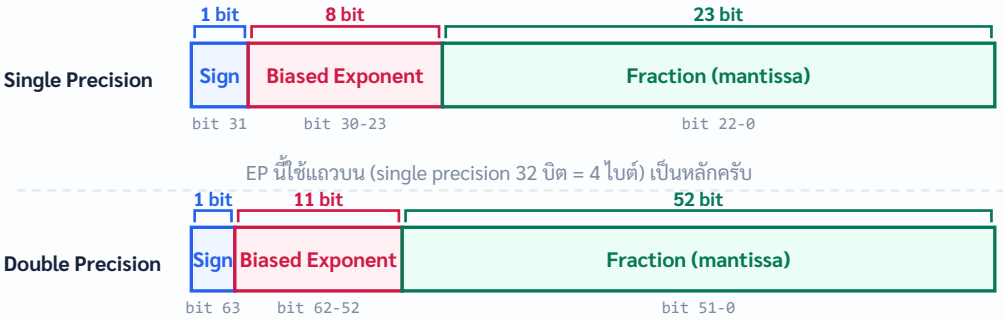
## 3 ชิ้นที่ว่านั้นคืออะไรบ้าง

| ชิ้นส่วน | ภาษาอังกฤษในสไลด์ | ทำหน้าที่อะไร |
|---|---|---|
| เครื่องหมาย | The sign-bit | บอกว่าบวกหรือลบ — **positive is '0' and negative is '1'** ใช้แค่ 1 บิตพอครับ |
| เลขชี้กำลัง | The biased exponent | บอกว่าให้เลื่อนจุดไปกี่ตำแหน่ง — คุมว่าเลขจะใหญ่หรือเล็ก (scale or range) |
| ตัวเลขนัยสำคัญ | The significand (fraction, mantissa) | ตัวเลขจริง ๆ ที่เหลือหลังจุด — คุมว่าละเอียดแค่ไหน (significant digits) |

**คำเดียวกัน 3 ชื่อ อย่าตกใจ**

ช่องสีเขียวช่องนั้นมีคนเรียก 3 ชื่อสลับกันไปมาครับ: `significand` · `fraction` · `mantissa` สไลด์อาจารย์เขียนว่า "The significand (fraction, mantissa)" คือบอกไว้เลยว่าเป็นตัวเดียวกัน ในชีทนี้พี่จะใช้คำว่า **mantissa** เป็นหลัก แต่ถ้าข้อสอบเขียนว่า fraction ก็คือช่องเดียวกันนี้ครับ

## หน้าตากล่องจริง ๆ




กล่อง 32 ช่อง (และ 64 ช่อง) แบ่งเป็น 3 ช่วงตายตัว — สีฟ้า = เครื่องหมาย · สีชมพู = เลขชี้กำลัง · สีเขียว = ตัวเลขนัยสำคัญ

**ตัวเลขที่ต้องจำให้ขึ้นใจ**

**Single precision · 32 บิต · 4 ไบต์**

$1 + 8 + 23 = 32$ · **bias = 127**

ในภาษา C คือชนิด `float` ครับ

**Double precision · 64 บิต · 8 ไบต์**

$1 + 11 + 52 = 64$ · **bias = 1023**

ในภาษา C คือชนิด `double` ครับ

**ทางลัดจำ bias**

bias ไม่ใช่เลขสุ่มครับ มันคือ **ครึ่งหนึ่งของช่วงที่กล่อง exponent เก็บได้ ลบ 1** พอดี

single: exponent 8 บิต เก็บได้ 0–255 → $2^{8-1} - 1 = 128 - 1 = \mathbf{127}$

double: exponent 11 บิต เก็บได้ 0–2047 → $2^{11-1} - 1 = 1024 - 1 = \mathbf{1023}$

เพราะฉะนั้นถ้าจำเลขไม่ได้ ให้นับบิต exponent แล้วคิดสูตร $2^{n-1} - 1$ ได้เลยครับ

กับดักแรกของ EP นี้

คนพลาดบ่อยมากคือ **นับบิตผิด** ครับ ช่อง mantissa มี **23** บิต ไม่ใช่ 24 (บิตที่ 24 คือ hidden bit ที่ไม่ได้เก็บ เดี๋ยวหัวข้อ 2 พี่จะอธิบาย) และช่อง exponent มี **8** บิต ไม่ใช่ 7 ถ้านับผิดแม้แต่บิตเดียว คำตอบฐานสิบหกจะเพี้ยนทั้งชุดครับ ตอนเขียนคำตอบให้ **นับซ้ำเสมอ**

## ศัพท์สอบของหัวข้อนี้

| English | แปลว่า | ข้อสอบเขียนมาแบบนี้ |
|---|---|---|
| floating-point number | เลขทศนิยมแบบจุดลอย | Express the following real numbers as floating points |
| single precision | ความละเอียดเดี่ยว (32 บิต) | ...as single-precision floating points |
| double precision | ความละเอียดคู่ (64 บิต) | ...using double precision |
| sign bit | บิตเครื่องหมาย | Determine the sign bit |
| biased exponent | เลขชี้กำลังที่บวก bias แล้ว | Calculate the biased exponent |
| significand / mantissa / fraction | ตัวเลขนัยสำคัญ (เศษหลังจุด) | the significand of the number |
| IEEE 754 standard | มาตรฐานเก็บเลขทศนิยม | using the IEEE 754 standard |
| real number | จำนวนจริง (มีเศษหลังจุดได้) | represent real numbers |

เทียบของเก่า

EP02 น้องเก็บ $-45$ ด้วยกล่อง 8 ช่องแบบ 2's complement — **ทั้งกล่องคือตัวเลขก้อนเดียว**
EP03 น้องจะเก็บ $-45.75$ ด้วยกล่อง 32 ช่อง แต่คราวนี้ **กล่องถูกซอยเป็น 3 ห้อง แต่ละห้องเก็บคนละหน้าที่**
นี่คือความต่างที่สำคัญที่สุดครับ — floating point **ไม่ได้** ใช้ 2's complement เก็บตัวเลขนะครับ มันใช้บิตเครื่องหมายแยกต่างหาก (sign-magnitude) เหมือนที่น้องเจอใน EP02 หัวข้อ sign-magnitude เป๊ะ

## 2 สามความลับของกล่อง: normalize · hidden bit · bias

**ตะขอเกี่ยว**

หัวข้อที่แล้วน้องรู้แล้วว่ากล่องมี 3 ห้อง คือ 1 + 8 + 23 บิตใช่ไหมครับ

งั้นสมมติว่าถ้าน้องเจอเลข $1101.11_2$ ละครับ น้องจะเอา “1101” ยัดเข้าห้องไหน แล้ว “11” หลังจุดยัดเข้าห้องไหน แล้วจุดหายไปไหน ถ้าตอบไม่ได้แปลว่ายังขาดกฎอีก 3 ข้อครับ — หัวข้อนี้คือกฎ 3 ข้อนั้น

### ความลับที่ 1 · normalize — บังคับให้เหลือ 1 ตัวหน้าจุดเสมอ

ปัญหาคือเลขก้อนเดียวกันเขียนได้หลายหน้าตาครับ ลองดูนี่:

$$1101.11_2 = 110.111 \times 2^1 = 11.0111 \times 2^2 = \mathbf{1.101111 \times 2^3}$$

ทุกบรรทัดคือเลขตัวเดียวกันหมด แค่เลื่อนจุดไปมา

ถ้าปล่อยให้เขียนได้หลายแบบ คอมพิวเตอร์จะเก็บยังไงก็ได้ แล้วเวลาเอามาเทียบกันจะเละครับ มาตรฐานเลยบังคับว่า **ต้องเลื่อนจุดจนเหลือเลข 1 ตัวเดียวหน้าจุด** เท่านั้น เรียกว่า `normalized form` ครับ

**เข้าเรื่อง · normalized form**

$$\text{Normalized form: } \mathbf{1.m \times 2^e}$$

**where “m” is the fraction and “e” is the exponent** (สไลด์หน้า 3 เขียนไว้แบบนี้เป๊ะครับ)

**ที่มาว่าทำไมต้องเป็น 1 เสมอ:** ในฐานสอง ตัวเลขหน้าจุดมีได้แค่ 0 กับ 1 ครับ ถ้าเลื่อนจุดแล้วได้ $0.\ldots$ แปลว่ายังเลื่อนไม่พอ ให้เลื่อนต่อ ถ้าได้ $11.\ldots$ แปลว่าเลื่อนน้อยไป ก็เลื่อนต่ออีก สุดท้ายมันจะไปหยุดที่ $1.\ldots$ เสมอ — **ฐานสองมีทางเดียวจริง ๆ** ครับ (ต่างจากฐานสิบที่ตัวหน้าจุดเป็น 1–9 ได้ทั้งหมด)

#### เลื่อนจุดแล้วเลขชี้กำลังเปลี่ยนยังไง




**เลื่อนซ้าย 3 ที → ต้องคูณกลับด้วย 2 ยกกำลัง 3 → e = 3**

12.75 → $1100.11_2$ → normalize เป็น $1.10011 \times 2^3$

**ทางลัด · นับตำแหน่งไม่ต้องคิดมาก**

ให้ดู**ส่วนจำนวนเต็ม**ครับ ถ้าส่วนจำนวนเต็มมี $k$ หลัก (โดยหลักซ้ายสุดเป็น 1) แล้ว $e = k - 1$ พอดีเลย

$1100.11$ → ส่วนจำนวนเต็มมี 4 หลัก → $e = 3$ · $1100011.00001$ → 7 หลัก → $e = 6$

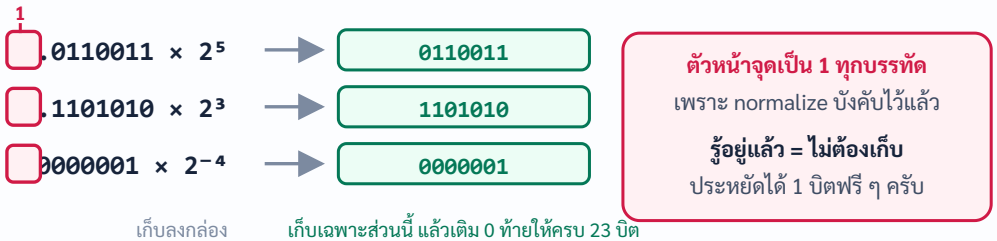
ไม่ต้องนั่งนับลูกศรทีละขั้นเลยครับ แต่ตอนเขียนคำตอบในข้อสอบ**ยังต้องวาดให้เห็นว่าเลื่อนกี่ที** เพราะคะแนนอยู่ที่ขั้นตอน

**กับดัก · เลขน้อยกว่า 1 ต้องเลื่อนขวา**

ถ้าเลขที่ได้เป็น $0.00101_2$ ตัวหน้าจุดเป็น 0 ครับ ต้องเลื่อนจุดไป**ทางขวา** จนเจอ 1 ตัวแรก → $1.01 \times 2^{-3}$
**เลื่อนขวา = เลขชี้กำลังติดลบ** ครับ นี่แหละคือเหตุผลที่ต้องมี bias ในความลับที่ 3

## ความลับที่ 2 · hidden bit — เลข 1 หน้าจุดไม่ต้องเก็บ

ทีนี้พี่ถามน้องหน่อยครับ ในเมื่อ normalize บังคับให้ตัวหน้าจุด**เป็น 1 ทุกครั้ง** น้องคิดว่ายังต้องเปลืองช่องเก็บมันไหมครับ




hidden bit (implied 1) — บิต 1 หน้าจุดไม่เคยถูกเก็บลงกล่อง

**เข้าเรื่อง · hidden bit (implied 1)**

**ไม่ต้องเก็บครับ** เพราะเรารู้อยู่แล้วว่ามันเป็น 1 เสมอ เวลาจะอ่านค่ากลับ ก็แค่เติม “1.” กลับเข้าไปหน้าเศษที่เก็บไว้

**ที่มาว่าได้อะไร:** ช่อง mantissa มี 23 ช่องจริง แต่พอเติม hidden bit กลับเข้าไป เราได้ความละเอียดเท่ากับ **24 บิต** คือได้บิตแถมมาฟรี ๆ 1 บิตครับ ไม่มีอะไรเสียหายเลย นี่คือความฉลาดของมาตรฐาน IEEE 754

**กับดักตัวจริง ข้อสอบเก็บคะแนนตรงนี้**

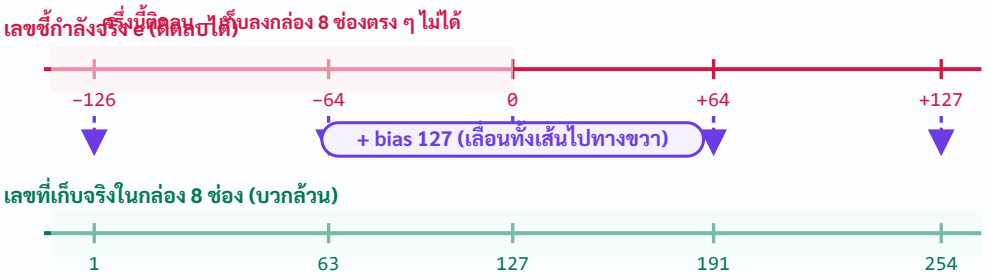
คนพลาดเยอะที่สุดของ EP นี้คือ **เผลอเก็บเลข 1 หน้าจุดลงไปด้วย** ครับ ถ้าน้องได้ $1.10001100001 \times 2^6$ แล้วเขียน mantissa เป็น `11000110000100000000` (ขึ้นต้นด้วย 1) — **ผิดครับ**
ที่ถูกคือตัดตัวหน้าจุดทิ้ง เหลือ `10001100001` แล้วเติม 0 ท้ายให้ครบ 23 บิต
ถ้าพลาดตรงนี้ เลขฐานสิบหกจะเพี้ยนทั้ง 8 ตัวเลยครับ

## ความลับที่ 3 · bias 127 — ทำไมต้องบวก 127

ตรงนี้คือคำถามที่น้องต้องตอบให้ได้ครับ ไม่ใช่ท่องว่า “บวก 127” เฉย ๆ

ปัญหาคือ เลขชี้กำลัง $e$ **ติดลบได้** ครับ (จากกับดักข้างบน: $0.00101_2 = 1.01 \times 2^{-3}$) แต่ช่อง exponent มีแค่ 8 ช่อง ถ้าจะเก็บเลขติดลบก็ต้องเสียไป 1 ช่องทำเป็นบิตเครื่องหมายอีก เหลือช่องเก็บค่าจริงแค่ 7 ช่อง — **เปลืองครับ** และเวลาเอามาเปรียบเทียบว่าเลขไหนใหญ่กว่ากันก็ยุ่งด้วย

ทางแก้ง่ายมากครับ: **เลื่อนเส้นจำนวนทั้งเส้นไปทางบวกให้หมด** โดยบวกค่าคงที่ตัวหนึ่งเข้าไปทุกครั้ง — ค่าคงที่ตัวนั้นเรียกว่า `bias`




bias 127 คือการ**เลื่อนเส้นจำนวนทั้งเส้น**ให้พ้นศูนย์ จะได้ไม่ต้องเปลืองบิตเครื่องหมายอีกตัวในช่อง exponent

**เข้าเรื่อง · biased exponent**

$$\text{biased exponent} = e + 127 \quad \text{(single precision)}$$

**Calculate the exponent: adding the bias to the exponent $e$ and convert the result to binary. For single-precision (32-bit), the bias is 127. For double-precision (64-bit), the bias is 1023.**

แปลเป็นภาษาพี่: หา $e$ จากการ normalize ก่อน แล้ว**บวก 127** แล้วค่อยแปลงผลลัพธ์เป็นฐานสอง 8 บิต — **ห้ามแปลง $e$ เป็นฐานสองก่อนแล้วค่อยบวกนะครับ** ให้บวกในฐานสิบให้เสร็จก่อน แล้วค่อยแปลง จะพลาดยากกว่ามาก

| $e$ จริง | $e + 127$ | เก็บลงกล่อง 8 บิต | หมายความว่า |
|---|---|---|---|
| $-3$ | 124 | **01111100** | เลขเล็กกว่า 1 เช่น 0.15625 |
| 0 | 127 | **01111111** | เลขอยู่ระหว่าง 1 ถึงเกือบ 2 |
| 3 | 130 | **10000010** | เลขประมาณ 8–16 เช่น 12.75 |
| 6 | 133 | **10000101** | เลขประมาณ 64–128 เช่น 99.03125 |

**ทางลัด · เช็คคำตอบด้วยสายตา**

ถ้าเลขที่โจทย์ให้มี**ขนาดมากกว่า 1** แล้ว biased exponent ที่น้องได้ **ต้องขึ้นต้นด้วย 1** เสมอครับ (เพราะ $e \geq 0 \rightarrow e + 127 \geq 127$ → บิตซ้ายสุดเป็น 1 เมื่อค่า $\geq 128$ หรือได้ `01111111` พอดีเมื่อ $e = 0$)
กลับกัน ถ้าเลขมีขนาด**น้อยกว่า 1** biased exponent จะขึ้นต้นด้วย `0111...` เสมอครับ ใช้เช็คเร็ว ๆ ก่อนส่งกระดาษได้เลย

**กับดัก · bias ไม่ใช่ 2's complement**

อย่าสับสนกับ EP02 นะครับ ช่อง exponent **ไม่ได้** ใช้ 2's complement ถ้าน้องเอา $e = -3$ ไปทำ 2's complement 8 บิต จะได้ `11111101` ซึ่ง**ผิด**
ที่ถูกคือ $-3 + 127 = 124 =$ `01111100`
จำง่าย ๆ ว่า **exponent ใช้ bias · ตัวเลขทั้งก้อนใช้ sign bit แยก · ไม่มี 2's complement ในกล่องนี้เลย**

## ศัพท์สอบของหัวข้อนี้

| English | แปลว่า | ข้อสอบเขียนมาแบบนี้ |
|---|---|---|
| normalize / normalized form | จัดให้เหลือ 1 ตัวหน้าจุด | Convert the number to binary and normalize it |

| English | แปลว่า | ข้อสอบเขียนมาแบบนี้ |
|---|---|---|
| bias | ค่าคงที่ที่บวกเข้ากับ exponent | the bias is 127 |
| hidden bit / implied 1 | บิต 1 หน้าจุดที่ไม่ได้เก็บ | the leading 1 is not stored |
| exponent | เลขชี้กำลัง | adding the bias to the exponent e |
| assemble | ประกอบเข้าด้วยกัน | Assemble the floating-point number |
| add zeros to the right | เติมศูนย์ทางขวา | (add zeros to the right if necessary) |

**เทียบของเก่า**

EP01 น้องแปลงเศษทศนิยมเป็นฐานสองด้วยวิธี **คูณ 2 เก็บตัวทด** ใช่ไหมครับ — ทักษะนั้นจะถูกใช้ในขั้นที่ 2 ของสูตรหน้าถัดไปเป๊ะ ๆ ไม่มีอะไรใหม่เลย

EP01 น้องยังจับ 4 บิตแปลงเป็นเลขฐานสิบหกได้ด้วย — ทักษะนั้นจะใช้ในขั้นสุดท้าย ตอนเขียนคำตอบ 32 บิตให้เป็นฐานสิบหก 8 ตัวครับ

# 3 สูตร 6 ขั้น: ฐานสิบ → single precision

**ตะขอเกี่ยว**

หัวข้อที่แล้วน้องรู้กฎครบ 3 ข้อแล้ว — normalize · hidden bit · bias ใช่ไหมครับ

งั้นสมมติว่าถ้าโจทย์ข้อสอบเขียนสั้น ๆ ว่า **"Convert $-118.625_{10}$ to floating point form using single precision"** ละครับ น้องจะเริ่มจากตรงไหน แล้วจะรู้ได้ยังไงว่าไม่ลืมขั้นไหน — หัวข้อนี้พี่จะให้**สูตรตายตัว 6 ขั้น** ที่ใช้ได้กับทุกข้อ ไม่ต้องคิดใหม่เลยครับ

**สูตรตายตัว · จำ 6 ขั้นนี้พอ**

สไลด์อาจารย์หน้า 3 เขียนไว้ 4 ขั้น แต่พี่ขอซอยขั้นที่ 2 กับ 4 ออกให้ละเอียดขึ้น เป็น **6 ขั้น** เพราะจุดที่คนพลาดมันซ่อนอยู่ในรอยต่อพอดีครับ

**1**

บวก `0` ลบ `1`

ดูแค่เครื่องหมาย

**หาบิตเครื่องหมาย (sign bit)**

**ทำอะไร:** ดูแค่ว่าโจทย์เป็นบวกหรือลบ — positive is '0' and negative is '1'

**ทำไมทำก่อน:** เพราะทำเสร็จใน 1 วินาที แล้ว**ตัดเครื่องหมายทิ้งไปเลย** ขั้นที่เหลือทั้งหมดคิดกับขนาด (magnitude) อย่างเดียวครับ จะได้ไม่ต้องลากเครื่องหมายไปด้วยตลอดทาง

**2**

ส่วนจำนวนเต็ม: หาร 2 เก็บเศษ — อ่านล่าง→บน

ส่วนทศนิยม: คูณ 2 เก็บตัวทด — อ่านบน→ล่าง

**แปลงเป็นฐานสอง — ทำ 2 ฝั่งแยกกัน**

**ทำอะไร:** ฝั่งซ้ายของจุด (ส่วนจำนวนเต็ม) ใช้ **หารสั้นด้วย 2 เก็บเศษ อ่านจากล่างขึ้นบน** · ฝั่งขวาของจุด (ส่วนทศนิยม) ใช้ **คูณ 2 เก็บตัวทด อ่านจากบนลงล่าง**

**ทำไมคนละทิศ:** ฝั่งซ้ายเศษตัวแรกที่ได้คือหลักที่มีน้ำหนักน้อยสุด ($2^0$) เลยต้องอ่านย้อน ส่วนฝั่งขวาตัวทดตัวแรกคือหลัก $2^{-1}$ ซึ่งใหญ่สุดของฝั่งนั้น เลยอ่านตามลำดับได้เลยครับ (อันนี้ EP01 สอนไปแล้ว)

**3**

`1100011.00001`

เลื่อนซ้าย 6 ตำแหน่ง

**normalize — เลื่อนจุดจนเหลือ 1.xxx**

**ทำอะไร:** เลื่อนจุดไปทางซ้ายจนเหลือเลข 1 ตัวเดียวหน้าจุด แล้ว**นับว่าเลื่อนไปกี่ตำแหน่ง**

**ทำไมต้องนับ:** จำนวนที่เลื่อนนั่นแหละคือ $e$ ครับ เลื่อนซ้าย $k$ ที = หารด้วย $2^k$ ไป $k$ ครั้ง ก็ต้องคูณกลับด้วย $2^k$ ค่าถึงจะเท่าเดิม → $e = +k$

ถ้าเลข**น้อยกว่า 1** ต้องเลื่อนขวา → $e$ ติดลบครับ

**4**

e + 127 = +

แล้วเขียนเป็นฐานสอง 8 บิต

**exponent = $e$ + bias 127 แล้วแปลงเป็นฐานสอง 8 บิต**

**ทำอะไร:** เอาจำนวนที่เลื่อนได้จากขั้น 3 บวก 127 **ในฐานสิบก่อน** แล้วค่อยแปลงเป็นฐานสอง

**ทำไมบวก 127:** เพราะ $e$ ติดลบได้ แต่กล่อง 8 ช่องไม่มีที่ให้บิตเครื่องหมาย การบวก 127 คือการเลื่อนเส้นจำนวนทั้งเส้นให้เป็นบวกหมด ($-127 \cdots +128$ กลายเป็น $0 \ldots 255$) จะได้ใช้ครบทั้ง 8 ช่องเก็บค่าจริง และเปรียบเทียบว่าเลขไหนใหญ่กว่ากันได้ตรง ๆ ครับ

**5**




**mantissa = เศษหลังจุด เติม 0 ให้ครบ 23 บิต**

**ทำอะไร:** เอาเฉพาะตัวเลข**หลังจุด**ของรูป normalize มาใส่ แล้วเติม 0 ทางขวาจนครบ 23 ช่อง ( add zeros to the right if necessary )

**ทำไมไม่เก็บ 1 หน้าจุด:** เพราะ normalize บังคับให้เป็น 1 อยู่แล้วทุกครั้ง รู้อยู่แล้วก็ไม่ต้องเปลืองช่องเก็บครับ — นี่คือ **hidden bit** เติม 0 ทางขวาไม่ทำให้ค่าเปลี่ยน เหมือน $1.5 = 1.50 = 1.500$ ในฐานสิบเลยครับ

**6**




ต่อกัน 32 บิต → จับ 4 บิต → ฐาน 16

**ประกอบ 32 บิต แล้วเขียนเป็นเลขฐานสิบหก**

**ทำอะไร:** ต่อกันตามลำดับ **sign** → **exponent** → **mantissa** ได้ 32 บิต แล้วจับกลุ่มละ 4 บิต จากซ้ายไปขวา แปลงเป็นฐานสิบหก 8 ตัว

**ทำไมต้องเป็นฐานสิบหก:** 32 บิตยาวเกินกว่าจะตรวจด้วยตาครับ พอเขียนเป็น 8 ตัวอักษร แล้วเทียบง่ายกว่ามาก — และข้อสอบมักขอคำตอบเป็นฐานสิบหกด้วย (จับ 4 บิตได้เป๊ะพอดี 8 กลุ่ม เพราะ $32 \div 4 = 8$ ครับ)

**ทางลัด · เช็ค 3 จุดก่อนส่งกระดาษ**

① นับ mantissa ว่าครบ **23** ตัวไหม (นับเป็นกลุ่มละ 5 จะเร็วกว่า) · ② exponent ครบ **8** ตัวไหม · ③ ทั้งแถบรวมกันได้ **32** ตัวไหม

สามข้อนี้ผ่านแล้วโอกาสผิดเหลือน้อยมากครับ

## I DO พี่ทำให้ดูครับ

**ตัวอย่างที่พี่ทำ** สไลด์ Week 2 หน้า 4

> Convert $-118.625_{10}$ to floating point form using single precision.

แปล: แปลง $-118.625$ ฐานสิบ ให้เป็นเลขทศนิยมแบบจุดลอย ความละเอียดเดี่ยว (32 บิต) ครับ — สไลด์เว้นว่างไว้ให้ทำเอง พี่ทำให้ดูครบทุกขั้นเลย

### ขั้น 1 — บิตเครื่องหมาย

โจทย์เป็นเลข**ติดลบ** ครับ → **sign bit = 1** แล้วตัดเครื่องหมายทิ้ง เหลือทำงานกับ $118.625$ อย่างเดียว

### ขั้น 2 — แปลง 118.625 เป็นฐานสอง (ทำ 2 ฝั่งแยกกัน)

**ขั้น 2a · ส่วนจำนวนเต็ม 118 — หารสั้นด้วย 2 เก็บเศษ**

```
118 / 2 = 59   R 0  <- LSB
 59 / 2 = 29   R 1
 29 / 2 = 14   R 1
 14 / 2 =  7   R 0
  7 / 2 =  3   R 1
  3 / 2 =  1   R 1
  1 / 2 =  0   R 1  <- MSB
```

อ่านล่างขึ้นบน = 1110110

**ขั้น 2b · ส่วนทศนิยม 0.625 — คูณ 2 เก็บตัวทด**

```
0.625 x 2 = 1.25   <- MSB
0.250 x 2 = 0.50
0.500 x 2 = 1.00   หยุด (เหลือ 0)
```

อ่านบนลงล่าง = 101

**รวมสองฝั่ง:** $118.625_{10} = \mathbf{1110110.101_2}$ — เอาผลฝั่งซ้ายไปไว้หน้าจุด ผลฝั่งขวาไว้หลังจุดครับ

**ตรวจเร็ว ๆ ตรงนี้ก่อนไปต่อ**

$1110110_2 = 64+32+16+0+4+2+0 = 118$ ✓ · $0.101_2 = 0.5+0+0.125 = 0.625$ ✓
ถ้าตรงนี้ผิด ที่เหลืออีก 4 ขั้นผิดหมดครับ เสียเวลา 10 วินาทีตรวจคุ้มมาก

### ขั้น 3 — normalize




**เลื่อนซ้าย 6 ที → ต้องคูณกลับด้วย 2 ยกกำลัง 6 → e = 6**

−118.625 → $1110110.101_2$ → normalize เป็น $1.110110101 \times 2^6$

ได้ $1.110110101 \times 2^6$ ครับ — เลื่อนจุดซ้าย 6 ตำแหน่ง เพราะฉะนั้น $\mathbf{e = 6}$

เช็คด้วยทางลัด: ส่วนจำนวนเต็ม 1110110 มี 7 หลัก → $e = 7 - 1 = 6$ ✓

### ขั้น 4 — exponent = $e$ + 127

$6 + 127 = 133$ → แปลง 133 เป็นฐานสอง 8 บิต:

**133 หารสั้นด้วย 2**

```
133 / 2 = 66   R 1  <- LSB
 66 / 2 = 33   R 0
 33 / 2 = 16   R 1
 16 / 2 =  8   R 0
  8 / 2 =  4   R 0
  4 / 2 =  2   R 0
  2 / 2 =  1   R 0
  1 / 2 =  0   R 1  <- MSB
-------------------------------
= 10000101   (ตรวจ: 128+4+1 = 133 ถูก)
```

**biased exponent = 10000101** ครบ 8 บิตพอดีครับ

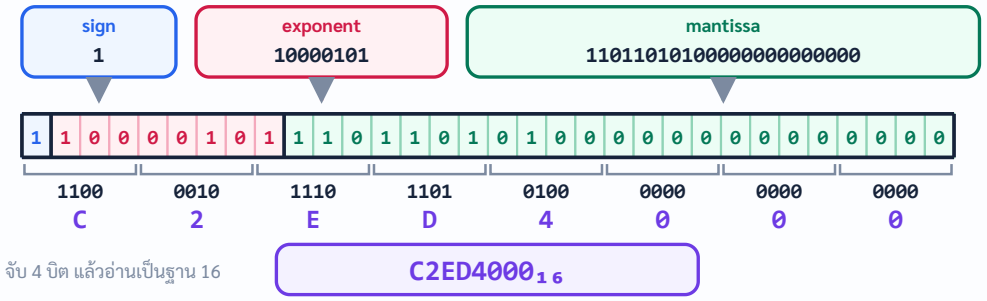
### ขั้น 5 — mantissa

จากรูป normalize $1.\underline{110110101} \times 2^6$ เอาเฉพาะส่วนขีดเส้นใต้ (หลังจุด) คือ `110110101` ซึ่งมี **9 บิต** ต้องเติม 0 อีก $23 - 9 = 14$ ตัวครับ

**นับให้ครบ 23 บิต (ทิ้งเลข 1 หน้าจุด = hidden bit)**

```
110110101 + 00000000000000
= 11011010100000000000000
  |----5----|----5----|----5----|----5---|
       ครบ 23 บิตพอดี
```

## ขั้น 6 — ประกอบแล้วแปลงเป็นฐานสิบหก




−118.625 ในรูป single precision — ตัวอย่างเดียวกับสไลด์หน้า 4

**คำตอบ**

$-118.625_{10}$ ในรูป single precision คือ

**1 10000101 11011010100000000000000**

หรือเขียนเป็นเลขฐานสิบหกได้ว่า $\mathbf{C2ED4000_{16}}$

**ตรวจคำตอบด้วยวิธีที่สอง — แกะกลับ**

อ่านจากแถบบิตกลับมาเป็นฐานสิบครับ:

sign = 1 → ติดลบ · exponent $= 10000101_2 = 133 \rightarrow e = 133 - 127 = 6$

mantissa → เติม hidden bit กลับ ได้ $1.110110101$

ค่า $= -1.110110101_2 \times 2^6 = -1110110.101_2 = -(118 + 0.625) = \mathbf{-118.625}$ ✓ ตรงกับโจทย์ครับ

# 4 ซ้อมมือ floating point (และวิธีอ่านย้อนกลับ)

## WE DO ทำด้วยกันครับ

**ทำด้วยกัน**

> Express $+12.75_{10}$ as a single-precision floating-point number. Give the answer in binary and in hexadecimal.

แปล: เขียน $+12.75$ เป็น single precision ตอบทั้งฐานสองและฐานสิบหกครับ คราวนี้เป็นเลข**บวก** ลองดูว่าอะไรเปลี่ยนไปบ้าง

**1**




**sign bit**

โจทย์เป็นบวก → sign = .....

**2**




**แปลง 12.75 เป็นฐานสอง**

ส่วนจำนวนเต็ม $12 =$ ..... · ส่วนทศนิยม $0.75 =$ .....

รวมได้ .....

**3**




**normalize**

เลื่อนจุดซ้าย ..... ตำแหน่ง → $1.$ ..... $\times 2^{?}$ → $e =$ .....

**4**




**exponent + bias**

$e + 127 =$ ..... → ฐานสอง 8 บิต = .....

**5**




**mantissa 23 บิต**

เศษหลังจุดคือ ..... ต้องเติม 0 อีก ..... ตัว

**6**




**ประกอบ + ฐานสิบหก**

32 บิต = ..... → ฐานสิบหก = .....

**เฉลย WE DO — เติมทุกช่องพร้อมเหตุผล**

**ขั้น 1 — sign = 0** เพราะโจทย์เป็นเลขบวกครับ กติกาคือ positive is '0' (นี่คือข้อเดียวที่ต่างจากตัวอย่าง $-118.625$ ของพี่ ที่เหลือทำเหมือนกันหมด)

**ขั้น 2 — แปลงเป็นฐานสอง** เพราะกล่องเก็บได้แต่ 0 กับ 1 เท่านั้น

**ส่วนจำนวนเต็ม 12 — หารสั้นด้วย 2**

```
12 / 2 = 6   R 0  <- LSB
 6 / 2 = 3   R 0
 3 / 2 = 1   R 1
 1 / 2 = 0   R 1  <- MSB
-----------------
อ่านล่างขึ้นบน = 1100
```

**ส่วนทศนิยม 0.75 — คูณ 2 เก็บตัวทด**

```
0.75 x 2 = 1.5   <- MSB
0.50 x 2 = 1.0   หยุด (เหลือ 0)
-----------------
อ่านบนลงล่าง = 11
```

รวมได้ $12.75_{10} = \mathbf{1100.11_2}$ ตรวจ: $8 + 4 = 12$ ✓ และ $0.5 + 0.25 = 0.75$ ✓

**ขั้น 3 — normalize เลื่อนซ้าย 3 ตำแหน่ง** เพราะต้องเหลือเลข 1 ตัวเดียวหน้าจุด จาก 1100.11 จุดต้องข้าม 1, 0, 0 มาอยู่หลังเลข 1 ตัวแรก = 3 ที ครับ




**เลื่อนซ้าย 3 ที → ต้องคูณกลับด้วย 2 ยกกำลัง 3 → e = 3**

12.75 → $1100.11_2$ → normalize เป็น $1.10011 \times 2^3$

ได้ $\mathbf{1.10011 \times 2^3} \rightarrow e = 3$ (ทางลัด: ส่วนจำนวนเต็ม 1100 มี 4 หลัก → $e = 4 - 1 = 3$ ✓)

**ขั้น 4 —** $e + 127 = 3 + 127 = 130$ ต้องบวก bias เพราะช่อง exponent เก็บเลขติดลบไม่ได้

แปลง 130 เป็นฐานสอง: $130 = 128 + 2 = 2^7 + 2^1 \rightarrow$ `10000010` (8 บิตพอดี)

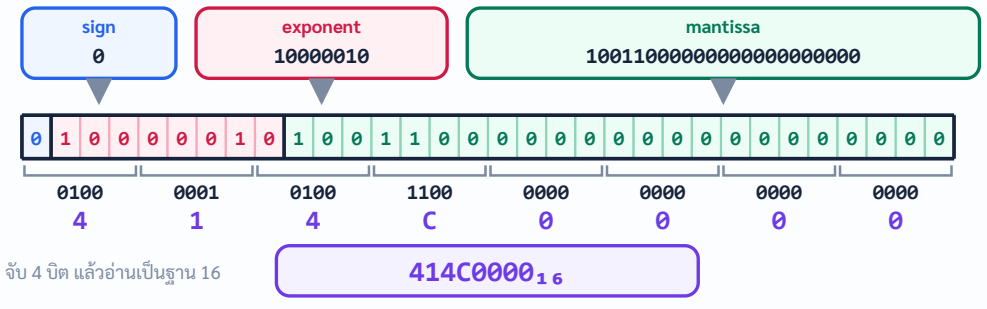
เช็คด้วยทางลัด: เลขมีขนาดมากกว่า 1 → exponent ต้องขึ้นต้นด้วย 1 ✓

**ขั้น 5 — mantissa** เศษหลังจุดของ $1.\underline{10011}$ คือ `10011` มี 5 บิต ต้องเติม 0 อีก $23 - 5 = \mathbf{18}$ ตัว

ได้ `10011000000000000000000`

เลข 1 หน้าจุด**ไม่เก็บ** ครับ เพราะเป็น hidden bit

**ขั้น 6 — ประกอบ** เรียง sign → exponent → mantissa




+12.75 ในรูป single precision

**ตอบ:** $+12.75_{10} = \mathbf{0\ 10000010\ 10011000000000000000000} = \mathbf{414C0000_{16}}$

*"+12.75 in single precision is 0 10000010 10011000000000000000000 = 414C0000 in hexadecimal."*

## อ่านย้อนกลับ: จาก 32 บิต → ฐานสิบ

**ตะขอเกี่ยว**

สมมติว่าถ้าข้อสอบให้แถบบิตมาแล้วถามว่า “เลขนี้คือเท่าไหร่” ละครับ ก็แค่**เดินสูตร 6 ขั้นถอยหลัง**ครับ ไม่มีอะไรใหม่เลย

**สูตรถอยหลัง 4 ขั้น**

① **ซอยแถบ** เป็น 1 / 8 / 23 บิต ② **อ่าน sign** 0 = บวก · 1 = ลบ ③ **exponent** แปลง 8 บิตเป็นฐานสิบ แล้ว**ลบ 127** ได้ $e$ ④ **เติม hidden bit** “1.” หน้า mantissa แล้วเลื่อนจุดไป $e$ ตำแหน่ง (บวก = เลื่อนขวา) แล้วอ่านค่าฐานสอง

**กับดักของทิศทางถอยหลัง**

ขั้น ③ ต้อง **ลบ 127** นะครับ ไม่ใช่บวก — คนพลาดตรงนี้เยอะมากเพราะจำแต่ว่า “127”

ตอนเข้ารหัส **บวก** 127 · ตอนถอดรหัส **ลบ** 127 — ตรงข้ามกันเสมอครับ

## YOU DO ตาน้องครับ

**Y4.1**

> Express $-40.25_{10}$ as a single-precision floating-point number. Give the answer in binary and in hexadecimal.

ใบ้: ทำตามสูตร 6 ขั้นเลยครับ ระวังตอนนับ 0 ที่ต้องเติมให้ครบ 23 บิต

**เฉลย Y4.1**

**ขั้น 1 — sign = 1** เพราะเป็นเลขติดลบครับ แล้วตัดเครื่องหมายทิ้ง เหลือคิดกับ $40.25$

**ขั้น 2 — แปลงเป็นฐานสอง** เพราะต้องเก็บลงกล่องที่มีแค่ 0 กับ 1

**ส่วนจำนวนเต็ม 40**

```
40 / 2 = 20   R 0  <- LSB
20 / 2 = 10   R 0
10 / 2 =  5   R 0
 5 / 2 =  2   R 1
 2 / 2 =  1   R 0
 1 / 2 =  0   R 1  <- MSB
= 101000   (32 + 8 = 40 ถูก)
```

**ส่วนทศนิยม 0.25**

```
0.25 x 2 = 0.5   <- MSB
0.50 x 2 = 1.0   หยุด
= 01   (0.25 ถูก)
```

รวมได้ $40.25_{10} = \mathbf{101000.01_2}$

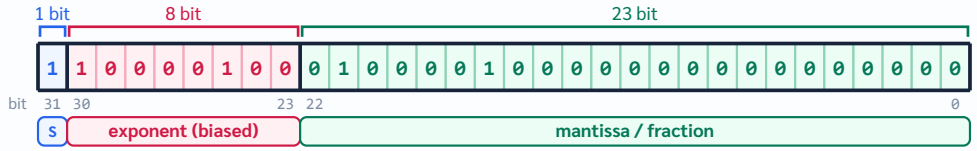
**ขั้น 3 — normalize** ส่วนจำนวนเต็ม $101000$ มี 6 หลัก ต้องเลื่อนจุดซ้าย **5** ตำแหน่ง จนเหลือ 1 ตัวเดียวหน้าจุด → $\mathbf{1.0100001 \times 2^5}$ → $e = 5$

สังเกตว่าเศษหลังจุดคือ $0100001$ — ตัว 0 ที่ตามหลัง 1 มาทันทีก็ต้องนับด้วยนะครับ ห้ามตัดทิ้ง

**ขั้น 4 — exponent** $5 + 127 = 132$ (บวก bias เพราะกล่อง exponent เก็บติดลบไม่ได้)
$132 = 128 + 4 = 2^7 + 2^2 \rightarrow$ `10000100` ครบ 8 บิต

**ขั้น 5 — mantissa** เศษหลังจุดคือ `0100001` มี 7 บิต เติม 0 อีก $23 - 7 = 16$ ตัว
ได้ `01000010000000000000000` (ทิ้งเลข 1 หน้าจุด — hidden bit)

**ขั้น 6 — ประกอบ 32 บิตแล้วจับ 4 บิต**




$-40.25_{10} = C2210000_{16}$

**ตรวจด้วยวิธีที่สอง — แกะกลับ:** exponent $10000100_2 = 132 \rightarrow e = 132 - 127 = 5$ · เติม hidden bit ได้ $1.0100001$ · เลื่อนขวา 5 ที่ $\rightarrow 101000.01_2 = 32 + 8 + 0.25 = 40.25$ · sign = 1 → ติดลบ → $-40.25$ ✓ ตรงกับโจทย์ครับ

**ตอบ: 1 10000100 01000010000000000000000 = $\mathbf{C2210000_{16}}$**
*"−40.25 in single precision is 1 10000100 01000010000000000000000 = $C2210000_{16}$."*

**Y4.2**

> Express $+5.5_{10}$ as a single-precision floating-point number.

ใบ้: ข้อนี้สั้น ใช้ตรวจว่าน้องนับ 0 เติมท้ายครบไหมครับ

**เฉลย Y4.2**

**ขั้น 1 — sign = 0** เพราะโจทย์เป็นบวกครับ

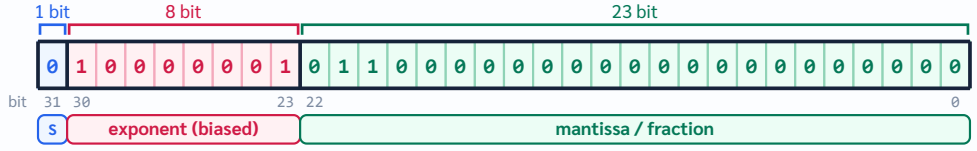
**ขั้น 2 — แปลงฐานสอง** $5 = 101_2$ (เพราะ $4 + 1 = 5$) และ $0.5 = 0.1_2$ (เพราะ $0.5 = 2^{-1}$ พอดี คูณ 2 รอบเดียวจบ)
รวมได้ $5.5_{10} = \mathbf{101.1_2}$

**ขั้น 3 — normalize** ส่วนจำนวนเต็ม 101 มี 3 หลัก → เลื่อนซ้าย **2** ตำแหน่ง → $\mathbf{1.011 \times 2^2} \rightarrow e = 2$

**ขั้น 4 — exponent** $2 + 127 = 129 = 128 + 1 \rightarrow$ `10000001`

**ขั้น 5 — mantissa** เศษหลังจุดคือ `011` มี 3 บิต เติม 0 อีก 20 ตัว → `01100000000000000000000`
ระวัง: ต้องเป็น **011** ไม่ใช่ **11** ครับ — เลข 0 ตัวแรกหลังจุดมีความหมาย ห้ามตัดทิ้ง

**ขั้น 6 — ประกอบ**




$+5.5_{10} = 40B00000_{16}$

**ตรวจด้วยวิธีที่สอง:** $e = 129 - 127 = 2 \cdot 1.011_2$ เลื่อนขวา 2 $\rightarrow 101.1_2 = 4 + 1 + 0.5 = 5.5$ ✓

**ตอบ: 0 10000001 01100000000000000000000 = $\mathbf{40B00000_{16}}$**

*"+5.5 in single precision is 0 10000001 01100000000000000000000 = $40B00000_{16}$."*

**Y4.3** ข้อยาก — exponent ติดลบ

> Express $-0.15625_{10}$ as a single-precision floating-point number.

ใบ้: ข้อนี้เลข**น้อยกว่า 1** ครับ จุดต้องเลื่อนไปทาง**ขวา** แล้ว $e$ จะติดลบ — นี่คือเหตุผลทั้งหมดที่ต้องมี bias 127 ลองดูว่าเกิดอะไรขึ้น

**เฉลย Y4.3**

**ขั้น 1 — sign = 1** เพราะติดลบครับ เหลือคิดกับ $0.15625$

**ขั้น 2 — แปลงฐานสอง** ส่วนจำนวนเต็มเป็น 0 อยู่แล้ว ทำแต่ฝั่งทศนิยม

**0.15625 คูณ 2 เก็บตัวทด**

```
0.15625 x 2 = 0.3125   <- MSB
0.31250 x 2 = 0.625
0.62500 x 2 = 1.25
0.25000 x 2 = 0.5
0.50000 x 2 = 1.0    หยุด
-------------------------------------------
= 0.00101   (0.125 + 0.03125 = 0.15625 ถูก)
```

**ขั้น 3 — normalize (คราวนี้เลื่อนขวา)** $0.00101_2$ ตัวหน้าจุดเป็น 0 ครับ ต้องเลื่อนจุด**ไปทางขวา** จนเจอเลข 1 ตัวแรก คือเลื่อน **3 ตำแหน่ง** → $\mathbf{1.01 \times 2^{-3}}$

**ทำไม $e$ ถึงติดลบ:** เลื่อนขวา 3 ที = คูณด้วย $2^3$ ไป 3 ครั้ง ต้องหารกลับด้วย $2^3$ ค่าถึงเท่าเดิม → คูณด้วย $2^{-3}$ → $e = -3$ ครับ

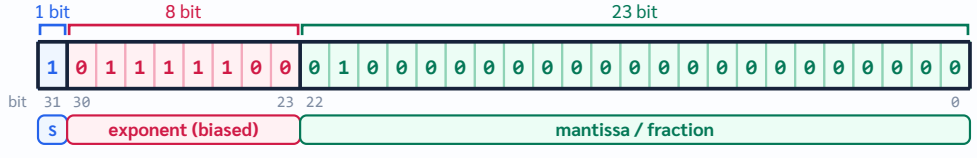
**ขั้น 4 — exponent $-3 + 127 = \mathbf{124}$ — นี่แหละคือประโยชน์ของ bias** ถ้าไม่มี bias น้องจะเก็บ $-3$ ลงกล่อง 8 ช่องไม่ได้เลยครับ

$124 = 64 + 32 + 16 + 8 + 4$ → **01111100**

เช็คด้วยทางลัด: เลขน้อยกว่า 1 → exponent ต้องขึ้นต้น **0111...** ✓

**ขั้น 5 — mantissa** เศษหลังจุดคือ **01** มี 2 บิต เติม 0 อีก 21 ตัว → **01000000000000000000000**

**ขั้น 6 — ประกอบ**




$-0.15625_{10} = BE200000_{16}$ — สังเกตว่า exponent เก็บ 01111100 = 124 ซึ่งแปลว่า e = 124 − 127 = −3

**ตรวจด้วยวิธีที่สอง:** exponent $01111100_2 = 124$ → $e = 124 - 127 = -3$ · เติม hidden bit ได้ $1.01$ · เลื่อน**ซ้าย** 3 ที่ (เพราะ $e$ ติดลบ) → $0.00101_2 = 0.125 + 0.03125 = 0.15625$ · sign = 1 → $-0.15625$ ✓

**ตอบ: 1 01111100 01000000000000000000000 = $\mathbf{BE200000_{16}}$**

*"−0.15625 in single precision is 1 01111100 01000000000000000000000 = $BE200000_{16}$."*

**Y4.4** ทิศทางถอยหลัง

> A single-precision floating-point number is stored as `0 10000011 01000010000000000000000`. What is its decimal value?

ใบ้: ใช้สูตรถอยหลัง 4 ขั้นครับ อย่าลืมว่าขั้น exponent ต้อง**ลบ** 127

**เฉลย Y4.4**

**ขั้น 1 — ซอยแถบเป็น 1 / 8 / 23** เพราะโครงกล่องตายตัวเสมอ:

sign = `0` · exponent = `10000011` · mantissa = `01000010000000000000000`

**ขั้น 2 — อ่าน sign** ได้ 0 → เลข**บวก** ครับ

**ขั้น 3 — หา** $e$ $10000011_2 = 128 + 2 + 1 = 131$ แล้ว**ลบ** 127 (ถอดรหัสต้องลบ) → $e = 131 - 127 = 4$

**ขั้น 4 — เติม hidden bit แล้วเลื่อนจุด** mantissa ที่มีความหมายคือ $0100001$ (0 ท้าย ๆ ตัดทิ้งได้)

เติม “1.” กลับเข้าไปหน้าสุด → $1.0100001 \times 2^4$

$e = +4$ → เลื่อนจุด**ขวา** 4 ตำแหน่ง → $10100.001_2$

**ขั้น 5 — อ่านค่าฐานสอง** เพราะยังต้องตอบเป็นฐานสิบครับ

$10100.001_2$

| 1 | 0 | 1 | 0 | 0 | . | 0 | 0 | 1 |
|---|---|---|---|---|---|---|---|---|
| $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ | | $2^{-1}$ | $2^{-2}$ | $2^{-3}$ |
| 16 | 0 | 4 | 0 | 0 | | 0 | 0 | 0.125 |

$16 + 4 + 0.125 = 20.125$ ครับ

**ตรวจด้วยวิธีที่สอง — เข้ารหัสกลับ:** $20.125 = 10100.001_2$ → normalize $1.0100001 \times 2^4$ → $e + 127 = 131 = 10000011$ ✓ · mantissa $= 0100001$ + 0 อีก 16 ตัว ✓ · sign บวก $= 0$ ✓ ตรงกับโจทย์ทุกบิตครับ

**ตอบ:** $\mathbf{+20.125_{10}}$

*“The stored value is +20.125 in decimal.”*

## 5 บวกเลขมีเครื่องหมาย — 4 กรณี

**ตะขอเกี่ยว**

EP02 น้องเขียน $-24$ เป็น `11101000` ได้แล้วใช่ไหมครับ

งั้นสมมติว่าถ้าน้องเจอโจทย์ $16 + (-24)$ ละครับ น้องต้องแยกคิดไหมว่า “อ๋อ ตัวหนึ่งบวก ตัวหนึ่งลบ งั้นต้องเอามาลบกัน แล้วดูว่าตัวไหนใหญ่กว่า”
ข่าวดีครับ — **ไม่ต้องเลย** วงจรบวกตัวเดียวจัดการได้หมดทั้ง 4 กรณี นี่คือเหตุผลทั้งหมดที่โลกนี้เลือกใช้ 2's complement ครับ

### ของใกล้ตัว: หน้าปัดนาฬิกา

ลองนึกภาพนาฬิกาบนโต๊ะน้องครับ ถ้าเข็มชี้ที่ 9 แล้วน้องอยากถอยกลับไป 3 ชั่วโมง น้องจะหมุน**ถอยหลัง** 3 ที ก็ได้ หรือหมุน**เดินหน้า** 9 ที ก็ได้ (เพราะ $12 - 3 = 9$) — ปลายทางเหมือนกันเป๊ะครับ

2's complement ทำงานแบบเดียวกันเลย: **“ลบ” ถูกแปลงเป็น “บวกไปอีกทาง”** เครื่องเลยไม่ต้องมีวงจรลบแยกต่างหาก มีแค่วงจรบวกอันเดียวพอครับ

**เข้าเรื่อง · กฎการบวก**

**Let's consider only the 2's complement form for representing signed numbers. The most widely used in computers and microprocessor-based systems.**

**The addition process is simple: Add the two numbers and discard any final carry bit.**

แปลเป็นภาษาพี่: **บวกกันตรง ๆ ทีละหลักเหมือนบวกเลขฐานสองปกติ แล้วถ้ามีตัวทดโผล่ออกมานอกกล่อง ก็ทิ้งไปซะ** — จบครับ ไม่ต้องดูเครื่องหมาย ไม่ต้องเทียบว่าใครใหญ่กว่า

**ทำไมทิ้งตัวทดตัวสุดท้ายได้**

กลับไปที่นาฬิกาครับ พอเข็มหมุนครบ 12 ชั่วโมง มันก็กลับมาที่เดิม — ส่วนที่ “ครบรอบ” ไม่มีความหมาย ทิ้งได้
กล่อง 8 บิตก็เหมือนกัน มันครบรอบทุก $2^8 = 256$ ตัวทดที่ล้นออกมาคือ “ครบรอบ 1 รอบ” พอดี **ทิ้งได้โดยที่คำตอบยังถูกอยู่**ครับ

### 4 กรณีที่สไลด์บอกไว้

| กรณี | สไลด์เขียนว่า | ผลลัพธ์ออกมาเป็นแบบไหน |
|---|---|---|
| **บวก + บวก** | **Addition of two positive numbers yields a positive number.** | ได้เลขบวก — **เสี่ยง overflow** |
| **บวก + ลบที่เล็กกว่า** | **Addition of a positive number and a smaller negative number yields a positive number.** | ได้เลขบวก — ปลอดภัยเสมอ |
| **บวก + ลบที่ใหญ่กว่า** | **Addition of a positive number and a larger negative number ... yields a negative number in 2's complement.** | ได้เลขลบในรูป 2's complement — ปลอดภัยเสมอ |
| **ลบ + ลบ** | **... or two negative numbers yields a negative number in 2's complement.** | ได้เลขลบ — **เสี่ยง overflow** |

**ทางลัด · อ่านค่าฐานสิบของผลลัพธ์**

พอบวกเสร็จ ให้ดู**บิตซ้ายสุด**ของคำตอบครับ
เป็น **0** → เป็นเลขบวก อ่านค่าตรง ๆ ได้เลย
เป็น **1** → เป็นเลขลบ ต้อง**ทำ 2's complement กลับ**ก่อน แล้วค่อยอ่านค่า แล้วเติมเครื่องหมายลบ
(อันนี้คือทักษะจาก EP02 เป๊ะ ๆ ครับ ไม่มีอะไรใหม่)

## I DO พี่ทำให้ดูครบทั้ง 4 กรณีครับ

**ตัวอย่างที่พี่ทำ** สไลด์ Week 2 หน้า 7

> Signed addition in 2's complement (8-bit) — work each one. Add the two numbers and discard any final carry bit. Give each answer in decimal as well.
> (a) 00000111 + 00000100 (b) 00001111 + 11111010
> (c) 00010000 + 11101000 (d) 11111011 + 11110111

แปล: บวกเลขมีเครื่องหมายแบบ 2's complement 8 บิต ตอบเป็นเลขฐานสิบด้วย — สังเกตว่าสี่ข้อนี้คือ 4 กรณีข้างบนพอดีเป๊ะครับ

### (a) 00000111 + 00000100 — บวก + บวก

**อ่านโจทย์ก่อน:** บิตซ้ายสุดของทั้งสองตัวเป็น 0 → เป็นเลขบวกทั้งคู่ $00000111_2 = 7$ และ $00000100_2 = 4$ ครับ

**(a) 00000111 + 00000100 · บวก + บวก**

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

**ผลลัพธ์ 00001011** บิตซ้ายสุดเป็น 0 → เลขบวก อ่านตรง ๆ ได้ $8 + 2 + 1 = 11$
**ตอบ: $00001011_2 = +11_{10}$** · ตรวจ: $7 + 4 = 11$ ✓
ไม่มีตัวทดออกนอกกล่อง และเครื่องหมายไม่พลิก → คำตอบใช้ได้ครับ

### (b) 00001111 + 11111010 — บวก + ลบที่เล็กกว่า

**อ่านโจทย์ก่อน:** ตัวแรกบิตซ้ายสุดเป็น 0 → $+15$ ตัวที่สองบิตซ้ายสุดเป็น **1** → เป็นเลขลบ ต้องแกะก่อน: 2's complement ของ $11111010$ คือ $00000110 = 6$ → เพราะฉะนั้นมันคือ $-\mathbf{6}$ ครับ

**(b) 00001111 + 11111010 · บวก + ลบที่เล็กกว่า**

```
  1 1 1 1 1
   0 0 0 0 1 1 1 1
+  1 1 1 1 1 0 1 0
-----------------
1  0 0 0 0 1 0 0 1
^ ตัวทดออกนอกกล่อง 8 บิต ทิ้งไป
```

**มีตัวทดออกนอกกล่อง 1 ตัว — ทิ้งครับ** เหลือ $00001001$
บิตซ้ายสุดเป็น 0 → เลขบวก อ่านได้ $8 + 1 = 9$

**ตอบ: $\mathbf{00001001_2 = +9_{10}}$** · ตรวจ: $15 + (-6) = 9$ ✓

จุดที่ต้องเก็บไว้ในใจ

ข้อนี้**มี**ตัวทดออกนอกกล่อง แต่คำตอบ**ถูก**ครับ — เก็บข้อสังเกตนี้ไว้ให้ดี เดี๋ยวหัวข้อ 6 เรื่อง overflow จะใช้

### (c) 00010000 + 11101000 — บวก + ลบที่ใหญ่กว่า

**อ่านโจทย์ก่อน:** $00010000_2 = +16$ · 11101000 บิตซ้ายสุดเป็น 1 → 2's complement กลับได้ $00011000 = 24$ → คือ $\mathbf{-24}$ ครับ

(c) 00010000 + 11101000 · บวก + ลบที่ใหญ่กว่า

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

**ผลลัพธ์ 11111000** ไม่มีตัวทดออกนอกกล่องเลย

บิตซ้ายสุดเป็น **1** → เป็นเลขลบ ต้องทำ 2's complement กลับเพื่ออ่านค่า: $11111000 \to 00000111 \to +1 \to 00001000 = 8$ → เพราะฉะนั้นคือ $-8$

**ตอบ: $\mathbf{11111000_2 = -8_{10}}$** · ตรวจ: $16 + (-24) = -8$ ✓

### (d) 11111011 + 11110111 — ลบ + ลบ

**อ่านโจทย์ก่อน:** ทั้งคู่บิตซ้ายสุดเป็น 1 → เป็นเลขลบทั้งคู่ $11111011 \to 00000101 = 5$ → คือ $-5$ · $11110111 \to 00001001 = 9$ → คือ $-9$ ครับ

(d) 11111011 + 11110111 · ลบ + ลบ

```
    1 1 1 1 1 1 1
   1 1 1 1 1 0 1 1
+  1 1 1 1 0 1 1 1
  ---------------
1  1 1 1 1 0 0 1 0
^ ตัวทดออกนอกกล่อง 8 บิต ทิ้งไป
```

**มีตัวทดออกนอกกล่อง — ทิ้ง** เหลือ 11110010

บิตซ้ายสุดเป็น 1 → เลขลบ ทำ 2's complement กลับ: $11110010 \to 00001101 \to +1 \to 00001110 = 14$ → คือ $-14$

**ตอบ: $\mathbf{11110010_2 = -14_{10}}$** · ตรวจ: $(-5) + (-9) = -14$ ✓

กับดักที่ทำให้เสียคะแนนฟรี

โจทย์บอกว่า “Give each answer in decimal as well” แปลว่า**ต้องตอบค่าฐานสิบด้วย**ครับ ไม่ใช่ตอบแค่แถบบิต และเวลาแปลงกลับ ถ้าบิตซ้ายสุดเป็น 1 แล้วน้อง**อ่านค่าตรง ๆ** (เช่นตอบ $11110010 = 242$) — **ผิดเต็ม ๆ** ครับ ต้องทำ 2's complement กลับก่อนเสมอ

## WE DO ทำด้วยกันครับ

**ทำด้วยกัน**

> Perform the following addition in the 2's complement form (8-bit): 00101101 + 00010011. Give the answer in binary and in decimal.

แปล: บวกแบบ 2's complement 8 บิต ตอบทั้งฐานสองและฐานสิบครับ

**1**

`0` 0 = บวก 1 = ลบ (ต้อง 2's comp กลับ)

อ่านค่าฐานสิบจากบิตซ้ายสุดก่อน

**อ่านเครื่องหมายของทั้งสองตัวก่อน**

00101101 บิตซ้ายสุด = ..... → ค่า .....

00010011 บิตซ้ายสุด = ..... → ค่า .....

**2**

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

ไล่จากขวาไปซ้าย

**บวกทีละหลักจากขวาไปซ้าย**

ผลลัพธ์ = ..... · มีตัวทดออกนอกกล่องไหม .....

**3**

`0` 0 = บวก 1 = ลบ (ต้อง 2's comp กลับ)

อ่านค่าฐานสิบจากบิตซ้ายสุดก่อน

**อ่านค่าฐานสิบของคำตอบ**

บิตซ้ายสุดของคำตอบ = ..... → ตอบ .....

**เฉลย WE DO — เติมทุกช่องพร้อมเหตุผล**

**ขั้น 1 — อ่านเครื่องหมายก่อนเสมอ** เพราะต้องรู้ว่ากำลังบวกอะไรกับอะไร จะได้ตรวจคำตอบตอนท้ายได้ครับ

00101101 บิตซ้ายสุด = **0** → เลขบวก อ่านตรง ๆ ได้ $32 + 8 + 4 + 1 = +\mathbf{45}$

00010011 บิตซ้ายสุด = **0** → เลขบวก อ่านตรง ๆ ได้ $16 + 2 + 1 = +\mathbf{19}$

เป็นกรณี **บวก + บวก** ครับ — กรณีนี้ต้องเช็ค overflow ตอนท้ายด้วย

**ขั้น 2 — บวกทีละหลัก** เพราะกฎบอกว่า “add the two numbers and discard any final carry bit”

**WE DO · 00101101 + 00010011**

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

ผลลัพธ์ = `01000000` · **ไม่มี**ตัวทดออกนอกกล่อง (carry out = 0) ครับ

**ขั้น 3 — อ่านค่าฐานสิบ** บิตซ้ายสุดของคำตอบ = **0** → เป็นเลขบวก อ่านตรง ๆ ได้เลย ไม่ต้องทำ 2's complement กลับ

$01000000_2 = 64$

**ตรวจ overflow:** บวกเลขบวกสองตัว ผลลัพธ์ออกมาเป็นบวก (บิตซ้ายสุด = 0) → **เครื่องหมายไม่พลิก = ไม่ overflow** คำตอบใช้ได้ครับ

**ตรวจด้วยวิธีที่สอง:** $45 + 19 = 64$ ✓ ตรงกันครับ

**ตอบ: $\mathbf{01000000_2 = +64_{10}}$ (valid, no overflow)**

*“00101101 + 00010011 = 01000000 = +64, and no overflow occurs.”*

## YOU DO ตาน้องครับ

**Y5.1**

> Perform the addition in the 2's complement form (8-bit): 00100000 + 11011000. Give the answer in binary and in decimal.

ใบ้: ดูบิตซ้ายสุดของตัวที่สองก่อนครับ ว่าเป็นบวกหรือลบ

**เฉลย Y5.1**

**ขั้น 1 — อ่านเครื่องหมาย** เพราะต้องรู้ว่าเป็นกรณีไหนใน 4 กรณี

$00100000$ บิตซ้ายสุด = 0 → บวก → $32 = +\mathbf{32}$

$11011000$ บิตซ้ายสุด = 1 → **ลบ** → ทำ 2's complement กลับ: $11011000 \rightarrow$ กลับบิต $00100111 \rightarrow +1 \rightarrow 00101000 = 40$ → คือ $-\mathbf{40}$

เป็นกรณี **บวก + ลบที่ใหญ่กว่า** → คาดว่าผลลัพธ์จะติดลบครับ

**ขั้น 2 — บวกทีละหลัก**

**Y5.1 · 00100000 + 11011000**

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

**ขั้น 3 — อ่านค่า** ไม่มีตัวทดออกนอกกล่อง ผลลัพธ์คือ `11111000`

บิตซ้ายสุด = **1** → เป็นเลข**ลบ** ต้องทำ 2's complement กลับก่อนอ่าน: $11111000 \rightarrow 00000111 \rightarrow +1 \rightarrow 00001000 = 8$ → คือ $-\mathbf{8}$

**ตรวจ overflow:** เครื่องหมายของตัวตั้งสองตัว**ต่างกัน** (บวกกับลบ) → กรณีนี้ **ไม่มีทาง overflow** เพราะผลลัพธ์ยิ่งเข้าใกล้ศูนย์ครับ

**ตรวจด้วยวิธีที่สอง:** $32 + (-40) = -8$ ✓

**ตอบ: $\mathbf{11111000_2 = -8_{10}}$ (valid)**

*“00100000 + 11011000 = 11111000 = −8 in decimal; no overflow.”*

**Y5.2**

> Perform the addition in the 2's complement form (8-bit): 11110000 + 11000000. Give the answer in binary and in decimal, and state whether the result is valid.

ใบ้: ทั้งคู่เป็นเลขลบครับ กรณีนี้**ต้อง**เช็ค overflow

**เฉลย Y5.2**

**ขั้น 1 — อ่านเครื่องหมาย**

$11110000$ บิตซ้ายสุด = 1 → ลบ → $11110000 \to 00001111 \to +1 \to 00010000 = 16$ → $\mathbf{-16}$

$11000000$ บิตซ้ายสุด = 1 → ลบ → $11000000 \to 00111111 \to +1 \to 01000000 = 64$ → $\mathbf{-64}$

เป็นกรณี **ลบ + ลบ** → กรณีเสี่ยง ต้องเช็ค overflow แน่นอนครับ

**ขั้น 2 — บวกทีละหลัก**

**Y5.2 · 11110000 + 11000000**

```
   1
   1 1 1 1 0 0 0 0
+  1 1 0 0 0 0 0 0
-----------------
1  1 0 1 1 0 0 0 0
^ ตัวทดออกนอกกล่อง 8 บิต ทิ้งไป
```

**ขั้น 3 — ทิ้งตัวทดแล้วอ่านค่า** มีตัวทดออกนอกกล่อง ทิ้งไป เหลือ `10110000`

บิตซ้ายสุด = 1 → ลบ → $10110000 \to 01001111 \to +1 \to 01010000 = 80$ → คือ $\mathbf{-80}$

**ขั้น 4 — ตรวจ overflow** ตัวตั้งทั้งสองเป็น**ลบ** (บิตซ้ายสุด 1 ทั้งคู่) ผลลัพธ์ก็เป็น**ลบ** (บิตซ้ายสุด 1) → **เครื่องหมายไม่พลิก = ไม่ overflow** คำตอบใช้ได้ครับ

(ต่อให้มีตัวทดออกนอกกล่องก็ไม่เกี่ยวเลยครับ — เดี๋ยวหัวข้อ 6 อธิบายเต็ม)

**ตรวจด้วยวิธีที่สอง:** $(-16) + (-64) = -80$ ✓ และ $-80$ ยังอยู่ในพิสัย $-128$ ถึง $+127$ ของ 8 บิต จึงเก็บได้จริง

**ตอบ: $\mathbf{10110000_2 = -80_{10}}$ (valid, no overflow)**

*“11110000 + 11000000 = 10110000 = −80; the sign bit stays 1, so the result is valid.”*

# 6 Overflow — เมื่อคำตอบใหญ่เกินกล่อง

**ตะขอเกี่ยว**

หัวข้อที่แล้วน้องสังเกตไหมครับว่า ข้อ (b) กับ (d) **มี**ตัวทดออกนอกกล่อง แต่คำตอบ**ถูก** ส่วนข้อ (a) กับ (c) **ไม่มี**ตัวทด แล้วคำตอบก็ถูกเหมือนกัน

งั้นสมมติว่าถ้าน้องเจอ $125 + 58$ ในกล่อง 8 บิตละครับ — คำตอบคือ 183 แต่กล่อง 8 บิตแบบมีเครื่องหมายเก็บได้แค่ถึง $+127$ เท่านั้น **แล้วเลข 183 มันจะไปโผล่เป็นอะไร** และเราจะรู้ได้ยังไงว่าคำตอบใช้ไม่ได้ — หัวข้อนี้ตอบครับ

## ของใกล้ตัว: แก้วน้ำที่รินเกินขอบ

ถ้าน้องรินน้ำใส่แก้วจนล้น น้ำที่ล้นออกมาไม่ได้หายไปเฉย ๆ ครับ มันไหลไปเลอะโต๊ะ แล้วปริมาณน้ำ**ในแก้ว**ที่น้องอ่านได้ก็ไม่ใช่ปริมาณที่น้องรินไปจริง ๆ อีกต่อไป

กล่อง 8 บิตก็เหมือนกันครับ ถ้าคำตอบใหญ่เกิน $+127$ หรือเล็กกว่า $-128$ ตัวเลขจะ “ล้น” ไปทับ**ช่องเครื่องหมาย** ทำให้เลขบวกกลายเป็นเลขลบไปดื้อ ๆ

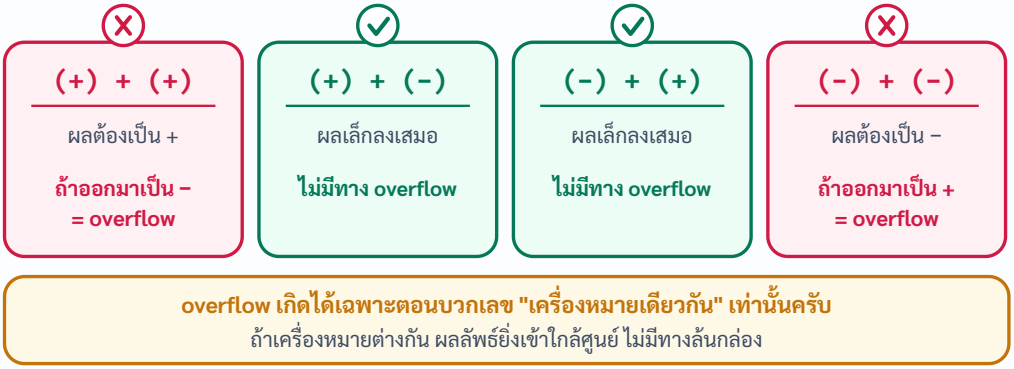
**เข้าเรื่อง · นิยาม overflow**

**When two numbers are added and the number of bits required to represent the sum exceeds the number of bits in the two numbers, an overflow results as indicated by an incorrect sign bit. Thus, the result is invalid.**

**Rule for detecting overflow: If the sign bit of the result is different than the sign bit of the numbers that are added, overflow is indicated.**

แปลเป็นภาษาพี่: ถ้าคำตอบต้องใช้บิตมากกว่าที่กล่องมี มันจะล้นไปทับบิตเครื่องหมาย ทำให้**เครื่องหมายผิด** — และนั่นคือสัญญาณเดียวที่เราใช้จับ overflow ครับ
**กฎสั้น ๆ ที่ต้องจำ: เครื่องหมายของตัวตั้งทั้งสองเหมือนกัน แต่เครื่องหมายของคำตอบพลิก = overflow**

## ทำไม overflow เกิดได้แค่ 2 กรณีจาก 4




4 กรณีของการบวกเลขมีเครื่องหมาย — สองกรณีกลางปลอดภัยเสมอ

**ที่มาของกฎ — ห้ามท่องลอย ๆ**

**กรณีเครื่องหมายต่างกัน (บวก+ลบ หรือ ลบ+บวก) ไม่มีทาง overflow เลยครับ** เพราะอะไร?
เพราะการเอาเลขบวกบวกกับเลขลบ คือการ**ทำให้ขนาดเล็กลง** ครับ ผลลัพธ์ต้องอยู่ระหว่างตัวตั้งสองตัวเสมอ ในเมื่อตัวตั้งทั้งสองอยู่ในกล่องได้อยู่แล้ว ผลลัพธ์ที่เล็กกว่านั้นก็ต้องอยู่ในกล่องได้ด้วย

**กรณีเครื่องหมายเหมือนกันถึงจะเสี่ยง** เพราะเป็นการ**ทำให้ขนาดใหญ่ขึ้น**
บวก+บวก: มากสุดคือ $127 + 127 = 254$ — เกิน $+127$ ไปเยอะครับ
ลบ+ลบ: น้อยสุดคือ $(-128) + (-128) = -256$ — เกิน $-128$ ไปเยอะเหมือนกัน

## ★ จุดตาย: ตัวทดออกนอกกล่อง $\neq$ overflow

ตรงนี้คือจุดที่คนพลาดมากที่สุดของทั้ง EP ครับ พี่ขอย้ำเป็นภาพ

```
01110000
+ 10101111
----------
1 00011111
```

carry out = 1
+112 + (−81) = +31
มีตัวทดออก แต่เครื่องหมายถูก → ไม่ใช่ overflow

ใช้ได้

```
01111101
+ 00111010
----------
10110111
```

carry out = 0
+125 + (+58) = +183 ?!
ไม่มีตัวทดออก แต่เครื่องหมายพลิก → overflow

OVERFLOW

ตัวทดออกนอกกล่อง กับ overflow เป็นคนละเรื่องกันครับ — ดูที่**เครื่องหมายของผลลัพธ์**เท่านั้น

**กับดักใหญ่ที่สุดของ EP นี้**

**อย่าเอา "มีตัวทดออกนอกกล่อง" ไปแปลว่า "overflow" เด็ดขาดครับ** — มันคนละเรื่องกันเลย
ดูตัวอย่าง (b) กับ (d) ในหัวข้อ 5 สิครับ ทั้งคู่มีตัวทดออก แต่คำตอบถูกทั้งคู่
ส่วนตัวอย่าง $125 + 58$ ข้างบน**ไม่มี**ตัวทดออกเลย แต่ overflow เต็ม ๆ
**สไลด์บอกไว้ชัดว่าให้ดูที่ sign bit อย่างเดียว** — "as indicated by an incorrect sign bit"

| เครื่องหมายตัวตั้ง | เครื่องหมายผลลัพธ์ | มีตัวทดออก? | overflow? | ตัวอย่างจากชีทนี้ |
|---|---|---|---|---|
| $+$ กับ $+$ | $+$ | ไม่ | ไม่ | (a) $7 + 4 = 11$ |
| $+$ กับ $-$ | $+$ | มี | ไม่ | (b) $15 + (-6) = 9$ |
| $+$ กับ $-$ | $-$ | ไม่ | ไม่ | (c) $16 + (-24) = -8$ |
| $-$ กับ $-$ | $-$ | มี | ไม่ | (d) $(-5) + (-9) = -14$ |
| $+$ กับ $+$ | $-$ | ไม่ | ★ใช่ | $125 + 58$ (สไลด์หน้า 9) |
| $-$ กับ $-$ | $+$ | มี | ★ใช่ | $(-100) + (-100)$ |

**ทางลัด · เช็ค overflow ใน 3 วินาที**

มองแค่ **3 บิต** ครับ — บิตซ้ายสุดของตัวตั้งที่ 1 · ตัวตั้งที่ 2 · และของคำตอบ
ถ้าสองตัวแรก**ต่างกัน** → ปลอดภัย ไม่ต้องดูต่อ
ถ้าสองตัวแรก**เหมือนกัน** → ดูตัวที่สาม ถ้าไม่เหมือนสองตัวแรก = **overflow**
อีกวิธีที่ให้ผลตรงกันเป๊ะ: **ตัวทดที่เข้าบิตเครื่องหมาย $\neq$ ตัวทดที่ออกจากบิตเครื่องหมาย = overflow** (ใช้ตรวจซ้ำได้ครับ)

**เทียบของเก่า**

EP02 น้องคำนวณ**พิสัย**ของเลข $n$ บิตแบบ 2's complement ได้ว่า $-2^{n-1}$ ถึง $+2^{n-1}-1$ ใช่ไหมครับ — สำหรับ 8 บิตคือ $-128$ ถึง $+127$
overflow ก็คือ**คำตอบหลุดออกนอกพิสัยนั้น**นั่นเองครับ ไม่ใช่เรื่องใหม่เลย เพียงแต่คราวนี้เรามีวิธีตรวจจับด้วยสายตาโดยไม่ต้องแปลงเป็นฐานสิบก่อน

## I DO พี่ทำให้ดูครับ

**ตัวอย่างที่พี่ทำ** — สไลด์ Week 2 หน้า 9

> 01111101 + 00111010 (8-bit signed) — is the answer valid? Check the sign bit of the result against the sign bits of the two operands.

แปล: บวกกันแล้วคำตอบใช้ได้ไหม ให้เทียบบิตเครื่องหมายของผลลัพธ์กับของตัวตั้งทั้งสองครับ

**1**

0 = บวก
1 = ลบ (ต้อง 2's comp กลับ)
อ่านค่าฐานสิบจากบิตซ้ายสุดก่อน

**อ่านเครื่องหมายตัวตั้งทั้งสองก่อน**

$01111101$ บิตซ้ายสุด = 0 → บวก → $64+32+16+8+4+1=+125$
$00111010$ บิตซ้ายสุด = 0 → บวก → $32+16+8+2=+58$
**เพราะ**ทั้งคู่เป็นบวกเหมือนกัน → นี่คือกรณี**เสี่ยง** ต้องเช็ค overflow แน่นอนครับ

**2**

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

ไล่จากขวาไปซ้าย

**บวกทีละหลักตามปกติ**

**เพราะ**วิธีบวกไม่เปลี่ยนเลยครับ overflow เป็นแค่การ**ตรวจสอบทีหลัง** ไม่ได้ทำให้ขั้นตอนบวกต่างไป

**3**

0 = บวก
1 = ลบ (ต้อง 2's comp กลับ)
อ่านค่าฐานสิบจากบิตซ้ายสุดก่อน

**เทียบบิตเครื่องหมาย 3 ตัว**

**เพราะ**กฎบอกว่า “if the sign bit of the result is different than the sign bit of the numbers that are added, overflow is indicated” — ดูแค่ 3 บิตนี้พอครับ

**01111101 + 00111010 · เช็ค overflow**

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

**ผลลัพธ์คือ 10110111** และ**ไม่มี**ตัวทดออกนอกกล่องเลย (carry out = 0)

**ถ้าน้องเชื่อคำตอบนี้จะได้อะไร**

ถ้าอ่านค่า 10110111 แบบเลขมีเครื่องหมาย: บิตซ้ายสุดเป็น 1 → เป็นลบ → 10110111 → 01001000 → +1 → 01001001 = 73 → คือ $-\mathbf{73}$

แต่คำตอบจริงคือ $125 + 58 = +\mathbf{183}$ ครับ — **คนละโลกกันเลย**

เลขบวกสองตัวบวกกันแล้วได้เลขลบ นี่คือสัญญาณ overflow ที่ชัดที่สุด

**คำตอบ**

บิตเครื่องหมายของตัวตั้ง = **0** และ **0** (บวกทั้งคู่) แต่บิตเครื่องหมายของผลลัพธ์ = **1** (ลบ) → **เครื่องหมายพลิก**

**ตอบ: OVERFLOW เกิดขึ้น ผลลัพธ์ 10110111 ใช้ไม่ได้ (invalid)**

**เหตุผล:** +183 เกินพิสัยสูงสุด +127 ของกล่อง 8 บิตแบบมีเครื่องหมาย ต้องขยายเป็น 16 บิตถึงจะเก็บได้ครับ

*“Overflow occurs: two positive operands produced a negative result, so the answer is invalid.”*

## WE DO ทำด้วยกันครับ

**ทำด้วยกัน**

> 01011010 + 01100100 (8-bit signed). Perform the addition and determine whether an overflow occurs.

แปล: บวกแล้วบอกว่าเกิด overflow ไหมครับ

**เฉลย WE DO**

**ขั้น 1 — อ่านเครื่องหมาย** เพราะต้องรู้ก่อนว่าเป็นกรณีเสี่ยงหรือไม่

01011010 บิตซ้ายสุด = 0 → บวก → $64 + 16 + 8 + 2 = +\mathbf{90}$

01100100 บิตซ้ายสุด = 0 → บวก → $64 + 32 + 4 = +\mathbf{100}$

**บวก + บวก = กรณีเสี่ยง** ครับ

**ขั้น 2 — ประมาณคำตอบไว้ในใจก่อน** $90 + 100 = 190$ ซึ่ง**มากกว่า 127** แล้ว → พี่เดาได้เลยว่าจะ overflow ครับ แต่ต้องแสดงวิธีให้เห็น

**ขั้น 3 — บวกทีละหลัก**

**WE DO overflow · 01011010 + 01100100**

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

**ขั้น 4 — เทียบบิตเครื่องหมาย** ตัวตั้ง = **0, 0** · ผลลัพธ์ 10111110 บิตซ้ายสุด = **1** → **ต่างกัน = overflow** ครับ

สังเกตว่า**ไม่มี**ตัวทดออกนอกกล่องเลย (carry out = 0) แต่ก็ยัง overflow — ตอกย้ำว่าตัวทดไม่ใช่ตัวชี้วัด

**ขั้น 5 — ถ้าหลงเชื่อจะได้อะไร** $10111110 \to 01000001 \to +1 \to 01000010 = 66$ → อ่านได้ $-66$ ซึ่งผิดจากคำตอบจริง $+190$ อย่างสิ้นเชิงครับ

**ตรวจด้วยวิธีที่สอง:** พิสัยของ 8 บิตแบบมีเครื่องหมายคือ $-128 \cdots +127 \cdot 190 > 127$ → หลุดพิสัย → ต้อง overflow ✓ ตรงกับที่ตรวจด้วยบิตเครื่องหมาย

**ตอบ: ผลลัพธ์** $10111110$ **— OVERFLOW เกิดขึ้น คำตอบใช้ไม่ได้**

*"Overflow occurs because two positive numbers (+90 and +100) produced a negative sign bit; +190 exceeds the 8-bit signed range of +127."*

## YOU DO ตาน้องครับ

**Y6.1**

> 01000000 + 01100000 (8-bit signed). Is the result valid? Explain.

ใบ้: บวก + บวก ครับ — กรณีเสี่ยง

**เฉลย Y6.1**

**ขั้น 1 — อ่านเครื่องหมาย** เพราะต้องรู้ว่าเป็นกรณีเสี่ยงไหม
$01000000$ บิตซ้ายสุด = 0 → บวก → $+\mathbf{64}$
$01100000$ บิตซ้ายสุด = 0 → บวก → $64 + 32 = +\mathbf{96}$
บวก + บวก → **กรณีเสี่ยง ต้องเช็ค**

**ขั้น 2 — บวก**

**Y6.1 · 01000000 + 01100000**

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

**ขั้น 3 — เทียบบิตเครื่องหมาย** ตัวตั้ง = **0, 0** (บวกทั้งคู่) ผลลัพธ์ $10100000$ บิตซ้ายสุด = **1** (ลบ) → เครื่องหมาย**พลิก** = **OVERFLOW** ครับ
และสังเกตว่าไม่มีตัวทดออกนอกกล่องเลย

**ขั้น 4 — ยืนยันด้วยฐานสิบ** $64 + 96 = 160$ ซึ่ง $> 127$ → หลุดพิสัยของ 8 บิตจริง
ถ้าหลงอ่านคำตอบ จะได้ $10100000 \to 01011111 \to +1 \to 01100000 = 96 \to -96$ ซึ่งผิดครับ

**ตอบ: ผลลัพธ์** $10100000$ **— invalid เพราะเกิด overflow**

*"The result is not valid: +64 and +96 are both positive but the sum has sign bit 1, so overflow occurred (+160 exceeds +127)."*

**Y6.2** ข้อหลอก

> 10000001 + 11111111 (8-bit signed). Is the result valid? Explain.

ใบ้: ข้อนี้**มี**ตัวทดออกนอกกล่อง อย่าเพิ่งรีบตอบว่า overflow นะครับ

**เฉลย Y6.2**

**ขั้น 1 — อ่านเครื่องหมาย**

$10000001$ บิตซ้ายสุด = 1 → ลบ → $10000001 \to 01111110 \to +1 \to 01111111 = 127 \to \mathbf{-127}$

$11111111$ บิตซ้ายสุด = 1 → ลบ → $11111111 \to 00000000 \to +1 \to 00000001 = 1 \to \mathbf{-1}$

ลบ + ลบ → กรณีเสี่ยง ต้องเช็คครับ

**ขั้น 2 — บวก**

**Y6.2 · 10000001 + 11111111**

```
   1 1 1 1 1 1 1
   1 0 0 0 0 0 0 1
+  1 1 1 1 1 1 1 1
-------------------
1  1 0 0 0 0 0 0 0
^ ตัวทดออกนอกกล่อง 8 บิต ทิ้งไป
```

**ขั้น 3 — เทียบบิตเครื่องหมาย** ตัวตั้ง = **1, 1** (ลบทั้งคู่) ผลลัพธ์ $10000000$ บิตซ้ายสุด = **1** (ลบ) → เครื่องหมาย**เหมือนเดิม ไม่พลิก** = **ไม่ overflow** ครับ

**ขั้น 4 — ตอบกับดักของข้อนี้** ข้อนี้**มี**ตัวทดออกนอกกล่อง (carry out = 1) ถ้าน้องใช้ตัวทดเป็นตัวตัดสิน น้องจะตอบผิดทันทีครับ — ตัวทดตัวนี้แค่ทิ้งไปเฉย ๆ

**ขั้น 5 — อ่านค่าคำตอบ** $10000000$ เป็นเลขลบ → $10000000 \to 01111111 \to +1 \to 10000000$

กรณีพิเศษครับ: $10000000$ ทำ 2's complement แล้วได้ตัวเดิม เพราะมันคือ $\mathbf{-128}$ พอดี ซึ่งเป็นค่าที่**น้อยที่สุด**ที่กล่อง 8 บิตเก็บได้

**ตรวจด้วยวิธีที่สอง:** $(-127) + (-1) = -128$ ✓ และ $-128$ ยัง**อยู่ในพิสัย** $-128 \cdots +127$ พอดีเป๊ะ → ไม่ overflow จริง ✓

ถ้าบวกอีกแค่ $-1$ เดียว ($-128 + (-1) = -129$) ถึงจะ overflow ครับ

**ตอบ: ผลลัพธ์** $10000000 = -128$ **— valid ไม่เกิด overflow**

*"Valid: both operands are negative and the result is also negative, so no overflow — the final carry bit is simply discarded."*

## 7 ลบ = บวกกับ 2's complement ของตัวลบ

ตะขอเกี่ยว

สองหัวข้อที่แล้วน้องบวกเลขมีเครื่องหมายได้แล้ว และตรวจ overflow เป็นแล้วใช่ไหมครับ

งั้นสมมติว่าถ้าน้องเจอโจทย์ที่เขียนว่า **ลบ** ละครับ เช่น $-25 - 19$ น้องต้องไปเรียนวิธีลบใหม่อีกชุดไหม
**ไม่ต้องเลยครับ** — วิชานี้ไม่มีวิธีลบ มีแต่วิธีบวก แล้วเราจะ**แปลงโจทย์ลบให้กลายเป็นโจทย์บวก** ก่อนลงมือทุกครั้ง

### ของใกล้ตัว: ลิ้นชักที่ดึงออกกับดันเข้า

ถ้าพี่บอกว่า “ดันลิ้นชักเข้าไป 5 เซนติเมตร” กับ “ดึงลิ้นชักออกมา $-5$ เซนติเมตร” มันคือคำสั่งเดียวกันเป๊ะครับ แค่พูดคนละแบบ

การลบก็เหมือนกันครับ — **ลบด้วย 19 = บวกด้วย** $-19$ และเราแปลง $+19$ ให้เป็น $-19$ ได้ด้วยการ**ทำ 2's complement** ซึ่งน้องทำเป็นแล้วจาก EP02

เข้าเรื่อง · กฎการลบ

**Subtraction is a special case of addition, for example, +9 − +6 = +9 + (−6). The sign of a positive or negative binary number can be changed by taking its 2's complement.**

**The subtraction process is: take the 2's complement of the subtrahend and add. Discard any final carry bit.**

แปลเป็นภาษาพี่ — **3 ขั้นตายตัว**:
① ทำ **2's complement ของตัวลบ** (subtrahend = ตัวที่อยู่หลังเครื่องหมายลบ)
② **บวก** เข้ากับตัวตั้ง (minuend) ตามปกติ
③ **ทิ้งตัวทดตัวสุดท้าย** ถ้ามี

กับดักคำศัพท์ที่ทำให้ทำผิดทั้งข้อ

$A - B \rightarrow$ minuend $= A$ (ตัวตั้ง) · subtrahend $= B$ (ตัวลบ)
**ทำ 2's complement เฉพาะตัวหลัง ($B$) เท่านั้นครับ** ห้ามทำตัวหน้าเด็ดขาด
ถ้าน้องเผลอทำ 2's complement ของตัวตั้ง น้องจะได้ $B - A$ ซึ่งเป็นคำตอบที่**เครื่องหมายกลับด้าน** — เสียคะแนนทั้งข้อโดยที่วิธีทำที่เหลือถูกหมดครับ

ทางลัด · วิธีทำ 2's complement ให้เร็วที่สุด

ไม่ต้องกลับบิตแล้วบวก 1 ทีละหลักก็ได้ครับ ใช้วิธี **“คัดลอกจนเจอ 1 ตัวแรก แล้วกลับที่เหลือ”**:
ไล่จาก**ขวาสุด**มาซ้าย — คัดลอก 0 มาเรื่อย ๆ จนเจอเลข **1** ตัวแรก ให้คัดลอก 1 ตัวนั้นมาด้วย จากนั้น**กลับค่าทุกบิตที่เหลือ**ทางซ้าย
เช่น 00010000 → คัดลอก 0000 แล้วเจอ 1 คัดลอกมาด้วยเป็น 10000 → กลับที่เหลือ 0001 → 1110 → ได้ **11110000** ตรงกับวิธียาวเป๊ะ แต่เร็วกว่ามากครับ

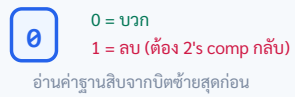
**I DO** พี่ทำให้ดูครับ

**ตัวอย่างที่พี่ทำ** สไลด์ Week 2 หน้า 11

> Subtract: −25 − 19 (8-bit 2's complement).
> minuend −25 = 11100111  subtrahend +19 = 00010011
> Take the 2's complement of the subtrahend and add. Remember: discard any final carry bit.

แปล: หาผลลบ $-25-19$ ในระบบ 2's complement 8 บิต สไลด์ให้แถบบิตมาให้แล้วทั้งสองตัว

**1**




**ระบุ minuend กับ subtrahend ให้ชัดก่อน**

**เพราะ**ถ้าสลับตัวจะได้คำตอบกลับด้านครับ

minuend (ตัวตั้ง) = $-25 = 11100111$ · subtrahend (ตัวลบ) = $+19 = 00010011$

ตรวจเร็ว ๆ: $11100111 \to 00011000 \to +1 \to 00011001 = 25$ ✓ เป็น $-25$ จริง

**2**




**ทำ 2's complement ของ subtrahend**

**เพราะ**เราต้องแปลง $+19$ ให้เป็น $-19$ ก่อน แล้วโจทย์ลบจะกลายเป็นโจทย์บวกครับ

**3**




**บวกแล้วทิ้งตัวทด**

**เพราะ**ตอนนี้มันคือโจทย์บวกธรรมดาแล้ว ใช้กฎเดิมจากหัวข้อ 5 ได้เลย

### ขั้น 2 — ทำ 2's complement ของ $+19$

**ทำ 2's complement ของตัวลบ +19 = 00010011**

```
   0 0 0 1 0 0 1 1   ตัวลบเดิม
   1 1 1 0 1 1 0 0   กลับทุกบิต (1's complement)
+  0 0 0 0 0 0 0 1   บวก 1
   ─────────────
   1 1 1 0 1 1 0 1   = 2's complement
```

ได้ 11101101 ครับ ตรวจว่าถูกไหม: $11101101 \to 00010010 \to +1 \to 00010011 = 19$ ✓ เพราะฉะนั้น 11101101 คือ $-\mathbf{19}$ จริง

### ขั้น 3 — บวก $-25 + (-19)$

**−25 + (−19) หลังทำ 2's complement แล้ว**

```
     1 1   1 1 1 1
   1 1 1 0 0 1 1 1
+  1 1 1 0 1 1 0 1
   ─────────────
1  1 1 0 1 0 1 0 0
^ ตัวทดออกนอกกล่อง 8 บิต ทิ้งไป
```

**มีตัวทดออกนอกกล่อง — ทิ้ง** เหลือ `11010100`

### ขั้น 4 — อ่านค่าฐานสิบ + ตรวจ overflow

บิตซ้ายสุดเป็น **1** → เลขลบ ต้องทำ 2's complement กลับ: $11010100 \to 00101011 \to +1 \to 00101100 = 32+8+4 = 44$ → คือ $-\mathbf{44}$

**ตรวจ overflow:** หลังแปลงเป็นโจทย์บวกแล้ว ตัวตั้งทั้งสองเป็น**ลบ** ($-25$ กับ $-19$) ผลลัพธ์ก็เป็น**ลบ** → เครื่องหมายไม่พลิก → **ไม่ overflow** ครับ

**คำตอบ**

$-25 - 19 = \mathbf{11010100_2} = \mathbf{-44_{10}}$ **(valid)**

ตรวจด้วยวิธีที่สอง: $-25 - 19 = -44$ ✓ และ $-44$ อยู่ในพิสัย $-128 \cdots +127$ ✓

*"Taking the 2's complement of +19 gives 11101101; adding to 11100111 gives 11010100 = −44."*

## WE DO ทำด้วยกันครับ

**ทำด้วยกัน**

> Perform the subtraction in the 2's complement form (8-bit): 01001000 − 00011010. Give the answer in binary and in decimal.

แปล: หาผลลบแบบ 2's complement 8 บิต ตอบทั้งฐานสองและฐานสิบครับ

**เฉลย WE DO — ทุกขั้นพร้อมเหตุผล**

**ขั้น 1 — อ่านโจทย์และระบุตัวตั้ง/ตัวลบ** เพราะทำ 2's complement ผิดตัวคือผิดทั้งข้อ

minuend = 01001000 บิตซ้ายสุด 0 → บวก → $64 + 8 = +\mathbf{72}$

subtrahend = 00011010 บิตซ้ายสุด 0 → บวก → $16 + 8 + 2 = +\mathbf{26}$

โจทย์คือ $72 - 26$ ครับ คาดว่าจะได้ $+46$

**ขั้น 2 — ทำ 2's complement ของตัวลบ** เพราะต้องแปลง $+26$ ให้เป็น $-26$

**ทำ 2's complement ของตัวลบ 00011010**

```
  0 0 0 1 1 0 1 0   ตัวลบเดิม
  1 1 1 0 0 1 0 1   กลับทุกบิต (1's complement)
+ 0 0 0 0 0 0 0 1   บวก 1
-------------------
  1 1 1 0 0 1 1 0   = 2's complement
```

ได้ **11100110** ครับ ตรวจ: $11100110 \to 00011001 \to +1 \to 00011010 = 26$ ✓

**ขั้น 3 — บวก** $72 + (-26)$ เพราะโจทย์ลบกลายเป็นโจทย์บวกแล้ว

**WE DO ลบ · 01001000 + 11100110 (2's comp ของ 00011010)**

```
  1
  0 1 0 0 1 0 0 0
+ 1 1 1 0 0 1 1 0
-----------------
1 0 0 1 0 1 1 1 0
^ ตัวทดออกนอกกล่อง 8 บิต ทิ้งไป
```

**ขั้น 4 — ทิ้งตัวทดแล้วอ่านค่า** มีตัวทดออกนอกกล่อง ทิ้งไป เหลือ `00101110`

บิตซ้ายสุดเป็น **0** → เลขบวก อ่านตรง ๆ ได้เลย: $32 + 8 + 4 + 2 = \mathbf{46}$

**ขั้น 5 — ตรวจ overflow** ตัวตั้งของการบวกคือ $+72$ (บวก) กับ $-26$ (ลบ) → เครื่องหมาย**ต่างกัน** → **ไม่มีทาง overflow** ครับ ไม่ต้องเช็คต่อ

**ตรวจด้วยวิธีที่สอง:** $72 - 26 = 46$ ✓ ตรงกันครับ

**ตอบ: $\mathbf{00101110_2 = +46_{10}}$ (valid)**
*“01001000 − 00011010 = 01001000 + 11100110 = 00101110 = +46 (final carry discarded).”*

## YOU DO ตาน้องครับ

**Y7.1**

> Perform the subtraction in the 2's complement form (8-bit): 00010010 − 01000101. Give the answer in binary and in decimal.

ใบ้: ข้อนี้ตัวตั้ง**เล็กกว่า**ตัวลบครับ คำตอบจะติดลบ — อย่าตกใจ วิธีทำเหมือนเดิมเป๊ะ

**เฉลย Y7.1**

**ขั้น 1 — ระบุตัวตั้ง/ตัวลบ** เพราะลำดับสำคัญมาก
minuend = $00010010 = 16 + 2 = +\mathbf{18}$ · subtrahend = $01000101 = 64 + 4 + 1 = +\mathbf{69}$
โจทย์คือ $18 - 69$ → คาดว่าได้ $-\mathbf{51}$ ครับ

**ขั้น 2 — ทำ 2's complement ของตัวลบ** $01000101$ เพื่อแปลง $+69$ เป็น $-69$

ทำ 2's complement ของตัวลบ 01000101

```
   0 1 0 0 0 1 0 1   ตัวลบเดิม
   1 0 1 1 1 0 1 0   กลับทุกบิต (1's complement)
+  0 0 0 0 0 0 0 1   บวก 1
   ---------------
   1 0 1 1 1 0 1 1   = 2's complement
```

ได้ **10111011** ตรวจ: $10111011 \to 01000100 \to +1 \to 01000101 = 69$ ✓

**ขั้น 3 — บวก** $18 + (-69)$

Y7.1 · 00010010 + 10111011 (2's comp ของ 01000101)

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

**ขั้น 4 — อ่านค่า** ข้อนี้**ไม่มี**ตัวทดออกนอกกล่อง ผลลัพธ์คือ `11001101`
บิตซ้ายสุดเป็น **1** → เลขลบ ต้องทำ 2's complement กลับก่อนอ่าน:
$11001101 \to$ กลับบิต $00110010 \to +1 \to 00110011 = 32 + 16 + 2 + 1 = 51$ → คือ $-\mathbf{51}$ ครับ

**ขั้น 5 — ตรวจ overflow** ตัวตั้งของการบวกคือ $+18$ กับ $-69$ → เครื่องหมาย**ต่างกัน** → **ไม่มีทาง overflow** ครับ

**ตรวจด้วยวิธีที่สอง:** $18 - 69 = -51$ ✓ และ $-51$ อยู่ในพิสัย $-128 \cdots + 127$ ✓

**ตอบ: $\mathbf{11001101_2 = -51_{10}}$ (valid)**

*“00010010 − 01000101 = 00010010 + 10111011 = 11001101 = −51 in decimal.”*

**ทางลัดตรวจงานของหัวข้อนี้**

ก่อนส่งกระดาษ ให้ถามตัวเอง 2 คำถามครับ

① ตัวตั้ง**ใหญ่กว่า**ตัวลบไหม → ถ้าใช่ คำตอบต้องเป็น**บวก** (บิตซ้ายสุด 0)

② ตัวตั้ง**เล็กกว่า**ตัวลบไหม → ถ้าใช่ คำตอบต้องเป็น**ลบ** (บิตซ้ายสุด 1)

ถ้าเครื่องหมายที่ได้ไม่ตรงกับที่คาด แปลว่าทำ 2's complement ผิดตัวครับ

# 8 คูณและหารเลขมีเครื่องหมาย

**ตะขอเกี่ยว**

ตอนนี้น้องบวกกับลบเลขมีเครื่องหมายได้แล้วใช่ไหมครับ

งั้นสมมติว่าถ้าน้องเจอ $0110 \times 1100$ ละครับ — **ตัวที่สองเป็นเลขลบ** น้องจะเอาแถบบิต 1100 ไปคูณตรง ๆ ได้ไหม

คำตอบคือ**ไม่ได้ครับ** และนั่นคือความต่างที่สำคัญที่สุดระหว่าง “บวก-ลบ” กับ “คูณ-หาร” ในระบบ 2's complement

**กฎเหล็กที่ต่างจากบวก-ลบ**

**When two binary numbers are multiplied, both numbers must be in true (uncomplemented) form. Thus, change any negative number to true form first.**

แปลเป็นภาษาพี่: การบวกกับการลบใช้แถบ 2's complement ได้เลย แต่**การคูณกับการหารต้องแปลงกลับเป็น “ขนาดจริง” (true form) ก่อนเสมอ** แล้วค่อยมาใส่เครื่องหมายให้ผลลัพธ์ตอนท้ายครับ

## ของใกล้ตัว: นับเหรียญกองเดียวกันหลายกอง

คอมพิวเตอร์ไม่มีวงจรคูณกับวงจรหารแยกต่างหากครับ — **คูณคือการบวกซ้ำ ๆ** (สไลด์เขียนว่า “accomplished using addition”) และ **หารคือการลบซ้ำ ๆ** (“accomplished using subtraction”) เหมือนน้องนับเหรียญกอง 5 บาท 4 กอง ก็แค่บวก 5 ไป 4 ครั้งครับ

## คูณ — วิธี partial products

**เข้าเรื่อง · 4 ขั้นของการคูณ**

① **ดูเครื่องหมายของทั้งสองตัวแล้วจดไว้** (ยังไม่ใช้ตอนนี้)
② **แปลงตัวที่ติดลบให้เป็น true form** ด้วยการทำ 2's complement
③ **คูณขนาดกันแบบ partial products** — starting with the LSB, each successive partial product is moved (shifted) one place to the left, and when all the partial products have been produced, they are added to get the final product
④ **ใส่เครื่องหมายให้ผลลัพธ์:** if the signs are the same, the product is positive; if the signs are different, the product is negative — ถ้าติดลบต้องทำ 2's complement ของผลคูณ

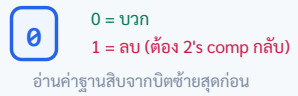
**I DO** พี่ทำให้ดูครับ

**ตัวอย่างที่พี่ทำ** สไลด์ Week 2 หน้า 14

> Multiply: 0110 × 1100 (give the product in 8 bits). Both numbers are 4-bit 2's complement. Put both numbers in true (uncomplemented) form first, then fix the sign of the product at the end.

แปล: คูณเลข 4 บิตแบบ 2's complement สองตัว ตอบผลคูณเป็น 8 บิตครับ

**1**




**อ่านเครื่องหมายทั้งสองตัวแล้วจดไว้**

$0110$ บิตซ้ายสุด = 0 → **บวก** → ค่า $4+2=+6$

$1100$ บิตซ้ายสุด = 1 → **ลบ** → ต้องแปลงก่อนถึงจะรู้ค่า

**เพราะ**เครื่องหมาย**ต่างกัน** → จดไว้เลยว่าผลคูณจะต้อง**ติดลบ**ครับ

**2**




**แปลงตัวลบเป็น true form**

$1100 \to$ กลับบิต $0011 \to +1 \to \mathbf{0100} = 4$ → เพราะฉะนั้น $1100 = -4$

**เพราะ**วงจรคูณทำงานกับ**ขนาด**เท่านั้น ถ้าเอาแถบ 2's complement ไปคูณตรง ๆ จะได้เลขมั่วครับ

**3**




**คูณขนาด $6 \times 4$ แบบ partial products**

**เพราะ**คูณในฐานสองง่ายมาก — บิตตัวคูณเป็น **1** ก็ลอกตัวตั้งลงมา เป็น **0** ก็เขียน 0 ทั้งแถว แล้วเลื่อนซ้ายทีละช่องทุกแถวครับ

**4**




**ใส่เครื่องหมายให้ผลคูณ**

**เพราะ**ขั้น 1 จดไว้ว่าเครื่องหมายต่างกัน → ผลคูณติดลบ → ต้องทำ **2's complement ของผลคูณ 8 บิต** ครับ

### ขั้น 3 — partial products ทุกแถว

**0110 × 0100 (true form ของ 1100) — partial products ทุกแถว**

```
          0 1 1 0
x         0 1 0 0
-----------------------------------
          0 0 0 0    bit 0 ของตัวคูณ = 0
        0 0 0 0 0    bit 1 ของตัวคูณ = 0
      0 1 1 0 0 0    bit 2 ของตัวคูณ = 1
    0 0 0 0 0 0 0    bit 3 ของตัวคูณ = 0
-----------------------------------
  0 0 0 1 1 0 0 0
```

ได้ขนาดของผลคูณ = `00011000` $= 16+8=24$ ครับ ตรวจ: $6 \times 4 = 24$ ✓

### ขั้น 4 — ใส่เครื่องหมายลบ

เครื่องหมายต่างกัน → ผลคูณต้องเป็นลบ → ทำ 2's complement ของ $00011000$:

$00011000 \to$ กลับบิต $11100111 \to +1 \to \mathbf{11101000}$

**คำตอบ**

$0110 \times 1100 = \mathbf{11101000_2} = \mathbf{-24_{10}}$ **(8 บิต)**

ตรวจด้วยวิธีที่สอง: $(+6) \times (-4) = -24$ ✓ · และแกะกลับ $11101000 \rightarrow 00010111 \rightarrow +1 \rightarrow 00011000 = 24$ → เป็น $-24$ จริง ✓

*“(+6) x (−4) = −24, stored as 11101000 in 8-bit 2's complement form.”*

**กับดักของการคูณ**

ถ้าน้องเอา $0110 \times 1100$ ไปคูณตรง ๆ โดยไม่แปลง true form ก่อน น้องจะได้ $6 \times 12 = 72 = 01001000$ ซึ่ง**ผิด**ครับ (คำตอบจริงคือ $-24$)

เพราะแถบ $1100$ ในระบบ 2's complement **ไม่ได้แปลว่า 12** แต่แปลว่า $-4$

อีกกับดักคือ**ลืมใส่เครื่องหมายตอนท้าย** — ตอบ $00011000$ เฉย ๆ ก็ผิดครึ่งข้อครับ

## WE DO ทำด้วยกันครับ

**ทำด้วยกัน**

> Multiply: 0101 × 0011 (4-bit 2's complement, give the product in 8 bits).

แปล: คูณเลข 4 บิต 2's complement สองตัว ตอบเป็น 8 บิตครับ

**เฉลย WE DO**

**ขั้น 1 — อ่านเครื่องหมาย** เพราะต้องรู้ว่าผลคูณจะบวกหรือลบ

$0101$ บิตซ้ายสุด = 0 → บวก → $4 + 1 = +\mathbf{5}$

$0011$ บิตซ้ายสุด = 0 → บวก → $2 + 1 = +\mathbf{3}$

**เครื่องหมายเหมือนกัน → ผลคูณเป็นบวก** ครับ

**ขั้น 2 — แปลง true form** ทั้งคู่เป็นบวกอยู่แล้ว → **ไม่ต้องแปลงอะไรเลย** (ขั้นนี้จำเป็นเฉพาะตอนมีเลขติดลบครับ)

**ขั้น 3 — partial products** ไล่จาก LSB ของตัวคูณขึ้นไป เลื่อนซ้ายทีละช่อง

**WE DO · 0101 × 0011 — partial products ทุกแถว**

```
            0 1 0 1
x           0 0 1 1
-------------------------------------
            0 1 0 1     bit 0 ของตัวคูณ = 1
          0 1 0 1 0     bit 1 ของตัวคูณ = 1
        0 0 0 0 0 0     bit 2 ของตัวคูณ = 0
      0 0 0 0 0 0 0     bit 3 ของตัวคูณ = 0
-------------------------------------
    0 0 0 0 1 1 1 1
```

**ขั้น 4 — ใส่เครื่องหมาย** เครื่องหมายเหมือนกัน → บวก → **ไม่ต้องทำ 2's complement** ใช้ผลลัพธ์ตรง ๆ ได้เลยครับ

**ตรวจด้วยวิธีที่สอง:** $00001111_2 = 8 + 4 + 2 + 1 = 15$ และ $5 \times 3 = 15$ ✓

**ตอบ:** $\mathbf{00001111_2 = +15_{10}}$

*“(+5) x (+3) = +15 = 00001111 in 8 bits; both signs are the same so the product is positive.”*

## หาร — ลบซ้ำแล้วนับจำนวนรอบ

เข้าเรื่อง · วิธีหารตามสไลด์

**The division operation in computers is accomplished using subtraction. When two binary numbers are divided, both numbers must be in true (uncomplemented) form.**

**Division process:**

① Subtract the divisor from the dividend using 2's complement addition to get the first partial remainder and add 1 to the quotient.

② Subtract the divisor from the partial remainder and add 1 to the quotient. If the result is positive, repeat until the result is zero or negative.

③ The quotient is the number of times that the divisor is subtracted.

แปลสั้น ๆ: **ลบตัวหารออกไปเรื่อย ๆ แล้วนับว่าลบได้กี่ครั้ง — จำนวนครั้งนั่นแหละคือผลหาร** ครับ

เครื่องหมายใช้กฎเดียวกับการคูณ: เหมือนกัน → บวก · ต่างกัน → ลบ (แล้วทำ 2's complement)

**ตัวอย่างที่พี่ทำ** สไลด์ Week 2 หน้า 17

> Divide: 01100100 ÷ 00011001 (8-bit). Subtract the divisor repeatedly using 2's complement addition. The quotient is the number of subtractions. Stop when the partial remainder reaches zero or goes negative.

แปล: หารเลข 8 บิต โดยลบตัวหารซ้ำ ๆ ด้วยการบวก 2's complement หยุดเมื่อเศษเหลือศูนย์หรือติดลบครับ

**ขั้น 1 — อ่านค่าและเครื่องหมาย** dividend $= 01100100 = 64 + 32 + 4 = 100$ (บวก) · divisor $= 00011001 = 16 + 8 + 1 = 25$ (บวก)

เครื่องหมายเหมือนกัน → ผลหารเป็น**บวก** และทั้งคู่เป็น true form อยู่แล้ว ไม่ต้องแปลงครับ

**ขั้น 2 — ทำ 2's complement ของตัวหารไว้ใช้ซ้ำ** $00011001 \rightarrow 11100110 \rightarrow +1 \rightarrow \mathbf{11100111}\ (= -25)$

ตัวนี้จะถูกบวกซ้ำ ๆ ทุกรอบครับ ทำครั้งเดียวพอ

**ขั้น 3 — ลบซ้ำแล้วนับรอบ** (ทุกรอบทิ้งตัวทดที่ออกนอกกล่อง)

**01100100 ÷ 00011001 — ลบซ้ำด้วย 2's complement addition**

```
2's complement of divisor 00011001 = 11100111

   0 1 1 0 0 1 0 0
+  1 1 1 0 0 1 1 1
-------------------
   0 1 0 0 1 0 1 1   เหลือ 75  ->  นับเป็นรอบที่ 1
   0 1 0 0 1 0 1 1
+  1 1 1 0 0 1 1 1
-------------------
   0 0 1 1 0 0 1 0   เหลือ 50  ->  นับเป็นรอบที่ 2
   0 0 1 1 0 0 1 0
+  1 1 1 0 0 1 1 1
-------------------
   0 0 0 1 1 0 0 1   เหลือ 25  ->  นับเป็นรอบที่ 3
   0 0 0 1 1 0 0 1
+  1 1 1 0 0 1 1 1
-------------------
   0 0 0 0 0 0 0 0   เหลือ 0  ->  นับเป็นรอบที่ 4
  ได้ศูนย์พอดี -> หยุด

quotient = 4  remainder = 25
```

**คำตอบ**

ลบได้ **4 ครั้ง** พอดีจนเหลือศูนย์ → **quotient** $= 4 = \mathbf{00000100_2}$ · **remainder** $= 0$

ตรวจด้วยวิธีที่สอง: $4 \times 25 = 100$ ✓ ตรงกับ dividend เป๊ะ

*“01100100 ÷ 00011001 = 00000100 (4) with remainder 0.”*

## YOU DO ตาน้องครับ

**Y8.1** สไลด์ Week 2 หน้า 18

> Divide: 00110010 ÷ 00001111 (8-bit). Subtract the divisor repeatedly using 2's complement addition, counting each subtraction. This one does not divide exactly — watch what happens on the last subtraction.

ใบ้: ข้อนี้หารไม่ลงตัวครับ ระวังตอนรอบสุดท้าย — รอบที่ทำให้ผลติดลบ **ไม่นับ** เป็นผลหาร

**เฉลย Y8.1**

**ขั้น 1 — อ่านค่าและเครื่องหมาย** เพราะต้องรู้ว่าต้องแปลง true form ไหม

dividend $= 00110010 = 32 + 16 + 2 = \mathbf{50}$ (บิตซ้ายสุด 0 → บวก)

divisor $= 00001111 = 8 + 4 + 2 + 1 = \mathbf{15}$ (บิตซ้ายสุด 0 → บวก)

เครื่องหมายเหมือนกัน → ผลหาร**บวก** · ทั้งคู่เป็น true form แล้ว ไม่ต้องแปลง

**ขั้น 2 — ทำ 2's complement ของตัวหารไว้ใช้ทุกรอบ**

$00001111 \rightarrow$ กลับบิต $11110000 \rightarrow +1 \rightarrow \mathbf{11110001}$ $(= -15)$

**ขั้น 3 — ลบซ้ำแล้วนับรอบ** เพราะ “the quotient is the number of times that the divisor is subtracted”

**00110010 ÷ 00001111 — ลบซ้ำ (ข้อนี้หารไม่ลงตัว)**

```
2's complement of divisor 00001111 = 11110001

   0 0 1 1 0 0 1 0
+  1 1 1 1 0 0 0 1
-------------------
   0 0 1 0 0 0 1 1   เหลือ 35  ->  นับเป็นรอบที่ 1
   0 0 1 0 0 0 1 1
+  1 1 1 1 0 0 0 1
-------------------
   0 0 0 1 0 1 0 0   เหลือ 20  ->  นับเป็นรอบที่ 2
   0 0 0 1 0 1 0 0
+  1 1 1 1 0 0 0 1
-------------------
   0 0 0 0 0 1 0 1   เหลือ 5  ->  นับเป็นรอบที่ 3
   0 0 0 0 0 1 0 1
+  1 1 1 1 0 0 0 1
-------------------
   1 1 1 1 0 1 1 0   ติดลบ (-10) -> หยุด ไม่นับรอบนี้

quotient = 3  remainder = 5
```

**ขั้น 4 — อ่านผลลัพธ์** รอบที่ 1, 2, 3 ได้ผลเป็นบวก → **นับทั้งสามรอบ**
รอบที่ 4 ได้ผลติดลบ ($-10$ ซึ่งขึ้นต้นด้วยบิต 1) → **หยุด และไม่นับรอบนี้** ครับ
เศษที่เหลือคือค่าก่อนรอบที่ทำให้ติดลบ = $00000101 = \mathbf{5}$

**ตรวจด้วยวิธีที่สอง:** $3 \times 15 + 5 = 45 + 5 = 50$ ✓ ตรงกับ dividend พอดีครับ

**ตอบ: quotient** $= 3 = \mathbf{00000011_2}$ **· remainder** $= 5 = \mathbf{00000101_2}$
*"00110010 ÷ 00001111 = 00000011 (3) with remainder 00000101 (5)."*

**กับดักของการหาร**

**อย่านับรอบที่ทำให้ผลติดลบ** ครับ — ถ้านับด้วยจะได้ 4 ซึ่งผิด ($4 \times 15 = 60 > 50$ ชัดเจนว่าเกิน)
วิธีดูว่าติดลบ: หลังบวกแล้ว**ไม่มีตัวทดออกนอกกล่อง** และบิตซ้ายสุดเป็น **1**
ส่วนรอบที่ยังบวกอยู่ จะ**มี**ตัวทดออกนอกกล่องทุกครั้ง (เพราะเรากำลังบวกเลขลบเข้ากับเลขบวกที่ใหญ่กว่า) — ใช้เป็นสัญญาณเตือนได้เลยครับ

## ศัพท์สอบของหัวข้อ 5–8

| English | แปลว่า | ข้อสอบเขียนมาแบบนี้ |
|---|---|---|
| discard any final carry bit | ทิ้งตัวทดตัวสุดท้าย | Add the two numbers and discard any final carry bit |
| overflow | ล้น (คำตอบเกินกล่อง) | an overflow results as indicated by an incorrect sign bit |
| valid / invalid | ใช้ได้ / ใช้ไม่ได้ | is the answer valid? |
| minuend | ตัวตั้ง (ตัวหน้าเครื่องหมายลบ) | minuend −25 = 11100111 |
| subtrahend | ตัวลบ (ตัวหลังเครื่องหมายลบ) | take the 2's complement of the subtrahend and add |
| true (uncomplemented) form | รูปขนาดจริง (ยังไม่ complement) | both numbers must be in true (uncomplemented) form |

| English | แปลว่า | ข้อสอบเขียนมาแบบนี้ |
| --- | --- | --- |
| partial product | ผลคูณย่อยแต่ละแถว | each successive partial product is moved one place to the left |
| partial remainder | เศษระหว่างทาง | subtract the divisor from the partial remainder |
| dividend / divisor / quotient | ตัวตั้งหาร / ตัวหาร / ผลหาร | the quotient is the number of subtractions |

STOP

# หน้าเฉลย Assignment 2

ตั้งแต่หน้านี้ไปคือเฉลย **Assignment #2 ข้อ 1–3** แบบละเอียดทุกข้อย่อย ครับ พี่ขอให้น้องลองทำเองก่อนนะครับ แล้วค่อยเปิดมาเทียบ การอ่านเฉลยแล้วรู้สึกว่า “อ๋อ ง่ายจัง” มันคนละเรื่องกับการทำได้เองในห้องสอบเลยครับ

## 9 เฉลย Assignment 2 · ข้อ 1

### ข้อ 1 — floating point

**ข้อ 1** Assignment #2

> Express the following real numbers as single-precision floating points: $-99.03125_{10}$

แปล: เขียน $-99.03125$ ฐานสิบ ให้อยู่ในรูปเลขทศนิยมแบบจุดลอย ความละเอียดเดี่ยว (32 บิต) ครับ

① โจทย์ซ่อนอะไรไว้

**ซ่อนไว้ 3 อย่างครับ**

① คำว่า single-precision แปลว่ากล่อง **32 บิต** และ bias คือ **127** (ถ้าเขียนว่า double ต้องใช้ 64 บิตกับ bias 1023 — คนละเลขกันเลย)

② เครื่องหมาย**ลบ**หน้าเลข → sign bit ต้องเป็น **1** (ข้อนี้ให้คะแนนฟรี อย่าลืม)

③ $0.03125$ ดูเหมือนเศษยาก ๆ แต่จริง ๆ มันคือ $\frac{1}{32} = 2^{-5}$ พอดี → แปลงเป็นฐานสองแล้วจะได้ $0.00001$ สั้น ๆ ไม่มีทศนิยมไม่รู้จบครับ

② เตรียมก่อนลงมือ

**ใช้วิธีไหน:** สูตร 6 ขั้นจากหัวข้อ 3 ครับ (sign → binary → normalize → exponent+bias → mantissa → assemble)

**ทำไมเลือกวิธีนี้:** เพราะเป็นวิธีเดียวที่สไลด์หน้า 3 กำหนดไว้ และข้อสอบให้คะแนนตาม**ขั้นตอน** ไม่ใช่แค่คำตอบสุดท้าย

**สูตร/กฎที่จะใช้:** · normalized form $= 1.m \times 2^{e}$ · biased exponent $= e + 127$ · mantissa = เศษหลังจุด เติม 0 ให้ครบ 23 บิต (ไม่เก็บ 1 หน้าจุด) · กล่อง $= 1 + 8 + 23$ บิต

### ③ ไล่ทีละขั้น · ขั้น 1 — บิตเครื่องหมาย

โจทย์คือ $-99.03125$ เป็นเลข**ติดลบ** ครับ ตามกฎ `positive is '0' and negative is '1'` → **sign bit** $= 1$
จากนี้ไปตัดเครื่องหมายทิ้ง ทำงานกับ $99.03125$ อย่างเดียวครับ

### ขั้น 2 — แปลง 99.03125 เป็นฐานสอง (แยก 2 ฝั่ง)

**ขั้น 2a · ส่วนจำนวนเต็ม 99 — หารสั้นด้วย 2 เก็บเศษ**

```
99 / 2 = 49   R 1  <- LSB
49 / 2 = 24   R 1
24 / 2 = 12   R 0
12 / 2 =  6   R 0
 6 / 2 =  3   R 0
 3 / 2 =  1   R 1
 1 / 2 =  0   R 1  <- MSB
```

อ่านล่างขึ้นบน = 1100011

**ขั้น 2b · ส่วนทศนิยม 0.03125 — คูณ 2 เก็บตัวทด**

```
0.03125 x 2 = 0.0625   <- MSB
0.06250 x 2 = 0.125
0.12500 x 2 = 0.25
0.25000 x 2 = 0.5
0.50000 x 2 = 1.0      หยุด
```

อ่านบนลงล่าง = 00001

**รวมสองฝั่ง:** $99.03125_{10} = \mathbf{1100011.00001_2}$

`ตรวจตรงนี้ก่อนไปต่อ (สำคัญมาก)`

$1100011_2 = 64 + 32 + 0 + 0 + 0 + 2 + 1 = 99$ ✓

$0.00001_2 = 2^{-5} = \frac{1}{32} = 0.03125$ ✓

ถ้าขั้นนี้ผิด อีก 4 ขั้นที่เหลือผิดหมดครับ — ตรวจ 10 วินาทีคุ้มมาก

### ขั้น 3 — normalize




**เลื่อนซ้าย 6 ที → ต้องคูณกลับด้วย 2 ยกกำลัง 6 → e = 6**

$-99.03125 \rightarrow 1100011.00001_2 \rightarrow$ normalize เป็น $1.10001100001 \times 2^6$

ต้องเลื่อนจุดไปทาง**ซ้าย 6 ตำแหน่ง** จนเหลือเลข 1 ตัวเดียวหน้าจุด
ได้ $1.10001100001 \times 2^6$ → เพราะฉะนั้น $e = 6$

เช็คด้วยทางลัด: ส่วนจำนวนเต็ม $1100011$ มี 7 หลัก → $e = 7 - 1 = 6$ ✓

### ขั้น 4 — exponent $= e + 127$

$6 + 127 = 133$ ครับ (บวก bias เพราะช่อง exponent เก็บเลขติดลบไม่ได้)

แปลง 133 เป็นฐานสอง 8 บิต

```
133 / 2 = 66   R 1  <- LSB
 66 / 2 = 33   R 0
 33 / 2 = 16   R 1
 16 / 2 =  8   R 0
  8 / 2 =  4   R 0
  4 / 2 =  2   R 0
  2 / 2 =  1   R 0
  1 / 2 =  0   R 1  <- MSB
-----------------------------
= 10000101     ตรวจ: 128 + 4 + 1 = 133 ถูก
```

**biased exponent** $= 10000101$ — 8 บิตพอดีครับ

เช็คด้วยทางลัด: เลขมีขนาดมากกว่า 1 → exponent ขึ้นต้นด้วย 1 ✓

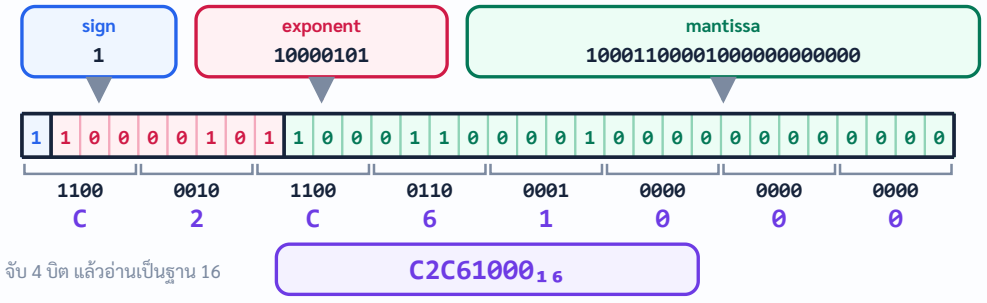
### ขั้น 5 — mantissa 23 บิต

จากรูป normalize $1.\underline{10001100001} \times 2^6$ เอาเฉพาะส่วนหลังจุด คือ `10001100001` ซึ่งมี **11 บิต**
ต้องเติม 0 ทางขวาอีก $23 - 11 = \mathbf{12}$ ตัวครับ

นับให้ครบ 23 บิต — ทิ้งเลข 1 หน้าจุด (hidden bit)

```
10001100001  +  000000000000
= 10001100001000000000000
  |---5---|---5---|---5---|---5---|---3-|
   11 บิตจริง + 12 ศูนย์ = 23 บิตพอดี
```

### ขั้น 6 — ประกอบ 32 บิตแล้วแปลงเป็นฐานสิบหก




−99.03125 ในรูป single precision — คำตอบ Assignment 2 ข้อ 1

**④ คำตอบสุดท้าย**

$-99.03125_{10}$ ในรูป single-precision floating point:

`1  10000101  10001100001000000000000`

(sign 1 บิต · exponent 8 บิต · mantissa 23 บิต — รวม 32 บิต)

เขียนติดกัน: `11000001011000110000100000000000`

เขียนเป็นฐานสิบหก: $\mathbf{C2C61000_{16}}$

**⑤ ตรวจคำตอบด้วยวิธีที่สอง — แกะกลับจากแถบบิต**

**ขั้น A — ซอยแถบ** $\rightarrow$ sign $= 1 \cdot$ exponent $= 10000101 \cdot$ mantissa $= 10001100001\ldots$

**ขั้น B — sign** $= 1 \rightarrow$ เลข**ติดลบ** ✓ ตรงกับโจทย์

**ขั้น C — exponent** $10000101_2 = 128 + 4 + 1 = 133$ แล้ว**ลบ** 127 (ถอดรหัสต้องลบ) $\rightarrow e = 133 - 127 = 6$ ✓

**ขั้น D — เติม hidden bit** "1." หน้า mantissa $\rightarrow 1.10001100001 \times 2^6$
เลื่อนจุดขวา 6 ตำแหน่ง $\rightarrow 1100011.00001_2$

**ขั้น E — อ่านค่าฐานสอง** $1100011_2 = 99$ และ $0.00001_2 = 0.03125$ $\rightarrow$ รวมเป็น $99.03125$
ใส่เครื่องหมายลบจาก sign bit $\rightarrow$ $\mathbf{-99.03125}$ ✓ **ตรงกับโจทย์ทุกหลักครับ**

**⑥ กับดักของข้อนี้ — 4 อย่างที่คนพลาดบ่อย**

**① เติม 0 ไม่ครบ 23 บิต** ถ้าน้องเติมแค่ 11 ตัว (นับผิดเป็น 22 บิต) จะได้ mantissa $10001100001000000000$ $\rightarrow$ ประกอบแล้วได้แค่ 31 บิต $\rightarrow$ เลขฐานสิบหกเพี้ยนทั้งชุดครับ
**วิธีกัน:** นับเป็นกลุ่มละ 5 $\rightarrow 5 + 5 + 5 + 5 + 3 = 23$

**② เผลอเก็บเลข 1 หน้าจุดลงไปด้วย** ถ้าเขียน mantissa เป็น `11000110000100000000` (ขึ้นต้น 1) $\rightarrow$ **ผิด**
คำตอบที่จะได้คือ `C2E30800` ซึ่งผิดจากคำตอบจริงครับ

**③ ลืมบวก bias หรือบวกผิดขั้นตอน** ถ้าเอา $e = 6$ ใส่ตรง ๆ เป็น $00000110$ จะได้ `1 00000110 10001100001...` $=$ `C3461000` ซึ่งผิด
และห้ามแปลง $e$ เป็นฐานสองก่อนแล้วค่อยบวก 127 นะครับ ให้บวกในฐานสิบให้เสร็จก่อนเสมอ

**④ แปลง** $0.03125$ **ผิด** คนมักเผลอเขียนเป็น $0.0001$ (4 ตำแหน่ง) ซึ่งคือ $0.0625$ ไม่ใช่ $0.03125$ ครับ
ต้อง **5 ตำแหน่ง** เพราะ $0.03125 = 2^{-5}$ — ถ้าพลาดตรงนี้ mantissa จะเลื่อนไปทั้งชุด

**หัวใจของข้อนี้**

**ข้อนี้วัดว่าน้องเดินสูตร 6 ขั้นครบไหม และนับบิตแม่นไหม — ไม่มีขั้นไหนยาก แต่พลาดขั้นเดียวคือผิดทั้งข้อครับ**

# 10 เฉลย Assignment 2 · ข้อ 2 (บวก)

**ข้อ 2** Assignment #2

> Perform addition in the 2's complement form:
> a. 00010110 + 00110011
> b. 01110000 + 10101111

แปล: หาผลบวกในระบบ 2's complement ครับ — สังเกตว่าโจทย์ให้มาเป็นแถบบิต 8 บิตทั้งคู่

**① โจทย์ซ่อนอะไรไว้ (ใช้กับทั้ง a และ b)**

**ซ่อนไว้ 3 อย่างครับ**

① คำว่า `in the 2's complement form` แปลว่าแถบบิตที่ให้มา**เป็นเลขมีเครื่องหมาย** → ต้องอ่านบิตซ้ายสุดเป็นเครื่องหมายก่อนเสมอ ห้ามอ่านเป็นเลขไม่มีเครื่องหมาย

② กล่องกว้าง **8 บิต** → คำตอบต้องเป็น 8 บิต และ**ตัวทดที่ล้นออกมาต้องทิ้ง**

③ โจทย์ไม่ได้พูดถึง overflow ตรง ๆ แต่**ต้องเช็คให้ทุกข้อ** เพราะถ้า overflow แล้วตอบคำตอบไปเฉย ๆ คือตอบผิด — โดยเฉพาะข้อ b ที่มีตัวทดออกนอกกล่อง

**② เตรียมก่อนลงมือ (ใช้กับทั้ง a และ b)**

**ใช้วิธีไหน:** บวกตรง ๆ ทีละหลักแล้วทิ้งตัวทดตัวสุดท้าย ตามสไลด์หน้า 6

**ทำไมเลือกวิธีนี้:** เพราะ 2's complement ออกแบบมาให้บวกได้เลยโดยไม่ต้องแยกกรณีเครื่องหมาย — วงจรบวกอันเดียวจัดการได้ทั้ง 4 กรณี

**กฎที่จะใช้:** · `Add the two numbers and discard any final carry bit` · บิตซ้ายสุดของคำตอบเป็น 1 → ต้องทำ 2's complement กลับก่อนอ่านค่าฐานสิบ · overflow เกิดเมื่อ**เครื่องหมายตัวตั้งเหมือนกันแต่เครื่องหมายผลลัพธ์พลิก**

## ข้อ 2a — 00010110 + 00110011

### ③ ไล่ทีละขั้น · ขั้น 1 — อ่านเครื่องหมายและค่าของตัวตั้งทั้งสอง

**ทำอะไร:** ดูบิตซ้ายสุดของแต่ละตัวก่อน **ทำไม:** เพราะต้องรู้ว่าเป็นกรณีไหนใน 4 กรณี และจะได้เอาไว้ตรวจ overflow ตอนท้ายครับ

00010110 — **บิตซ้ายสุด = 0 → บวก**

| 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| 0 | 0 | 0 | 16 | 0 | 4 | 2 | 0 |

00110011 — **บิตซ้ายสุด = 0 → บวก**

| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|
| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| 0 | 0 | 32 | 16 | 0 | 0 | 2 | 1 |

$00010110_2 = 16 + 4 + 2 = +\mathbf{22}$ · $00110011_2 = 32 + 16 + 2 + 1 = +\mathbf{51}$

ทั้งคู่**เป็นบวก** → กรณี **บวก + บวก** → เป็นกรณี**เสี่ยง overflow** ต้องเช็คตอนท้ายครับ

### ขั้น 2 — บวกทีละหลักจากขวาไปซ้าย

**ทำอะไร:** บวกคอลัมน์ต่อคอลัมน์พร้อมตัวทด **ทำไม:** เพราะกฎบอกว่า “add the two numbers” ไม่ต้องแปลงอะไรก่อนเลย

ข้อ 2a · 00010110 + 00110011

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

ผลลัพธ์ = **01001001** · **ไม่มี**ตัวทดออกนอกกล่อง (carry out = 0) จึงไม่มีอะไรต้องทิ้งครับ

### ขั้น 3 — อ่านค่าฐานสิบของคำตอบ

**ทำอะไร:** ดูบิตซ้ายสุดของคำตอบก่อน **ทำไม:** เพราะถ้าเป็น 1 ต้องทำ 2's complement กลับก่อนอ่าน

คำตอบ 01001001 — บิตซ้ายสุด = 0 → เป็นบวก อ่านตรง ๆ ได้เลย

| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
|---|---|---|---|---|---|---|---|
| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| 0 | 64 | 0 | 0 | 8 | 0 | 0 | 1 |

$64 + 8 + 1 = \mathbf{73}$ ครับ

### ขั้น 4 — ตรวจ overflow

บิตเครื่องหมายของตัวตั้ง = **0** และ **0** (บวกทั้งคู่) · บิตเครื่องหมายของผลลัพธ์ = **0** (บวก)
→ **เครื่องหมายไม่พลิก = ไม่เกิด overflow** คำตอบใช้ได้ครับ

ยืนยันอีกทาง: 73 อยู่ในพิสัย $-128 \cdots +127$ ของ 8 บิต ✓

**④ คำตอบข้อ 2a**

**00010110 + 00110011 = 01001001**

ค่าฐานสิบ: $+\mathbf{22} + (+\mathbf{51}) = +\mathbf{73}_{10}$ · **ไม่มี overflow (valid)**

**⑤ ตรวจคำตอบด้วยวิธีที่สอง**

**วิธีที่ 1 — แปลงเป็นฐานสิบแล้วบวก:** $22 + 51 = 73$ ✓ ตรงกับ $01001001_2 = 73$
**วิธีที่ 2 — ลบกลับ:** $01001001 - 00110011$ ควรได้ 00010110 กลับมา → $73 - 51 = 22$ ✓
**วิธีที่ 3 — เช็คบิตคู่/คี่:** 22 เป็นเลขคู่ (บิตขวาสุด 0) + 51 เป็นเลขคี่ (บิตขวาสุด 1) → ผลบวกต้องเป็น**คี่** → บิตขวาสุดของคำตอบต้องเป็น 1 ✓ (01001001 ลงท้ายด้วย 1)

**⑥ กับดักของข้อ 2a**

**กับดักคือ "ง่ายจนเผลอ"** ครับ — ข้อนี้เป็นบวกทั้งคู่ ไม่มีตัวทดล้น หลายคนเลย**ข้ามการเช็ค overflow** ไปเลย
แต่ข้อสอบวิชานี้มักหักคะแนนถ้าไม่แสดงการตรวจสอบครับ — ต้องเขียนกำกับไว้เสมอว่า *"both operands positive, result positive → no overflow."*
อีกกับดักคือ**บวกตัวทดพลาดที่หลัก** $2^1$ — ตรงนั้น $1 + 1 = 10$ ต้องเขียน 0 ทดขึ้น 1 ถ้าเผลอเขียน 1 จะได้ 01001011 = 75 ซึ่งผิดครับ

## ข้อ 2b — 01110000 + 10101111 (ข้อที่ต้องเช็ค overflow ให้ดี)

### ③ ไล่ทีละขั้น · ขั้น 1 — อ่านเครื่องหมายและค่าของตัวตั้งทั้งสอง

**ทำอะไร:** ดูบิตซ้ายสุด **ทำไม:** ข้อนี้ตัวที่สองขึ้นต้นด้วย **1** ซึ่งแปลว่าเป็น**เลขลบ** ถ้าอ่านผิดเป็น 175 คือผิดทั้งข้อครับ

**ตัวที่หนึ่ง** $01110000$ **— บิตซ้ายสุด = 0 → บวก**

| 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|
| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| 0 | 64 | 32 | 16 | 0 | 0 | 0 | 0 |

$01110000_2 = 64 + 32 + 16 = +\mathbf{112}$

ส่วนตัวที่สอง $10101111$ บิตซ้ายสุดเป็น **1** → เป็น**เลขลบ** ต้องทำ 2's complement กลับเพื่อหาขนาดครับ:

**หาขนาดของ 10101111 — ทำ 2's complement กลับ**

```
   1 0 1 0 1 1 1 1   แถบเดิม (เป็นเลขลบ)
   0 1 0 1 0 0 0 0   กลับทุกบิต
+  0 0 0 0 0 0 0 1   บวก 1
   ---------------
   0 1 0 1 0 0 0 1   = 64 + 16 + 1 = 81
```

เพราะฉะนั้น $10101111_2 = -\mathbf{81}$ ครับ
โจทย์ข้อนี้จริง ๆ คือ $(+\mathbf{112}) + (-\mathbf{81})$ → กรณี **บวก + ลบที่เล็กกว่า** → คาดว่าผลลัพธ์จะเป็น**บวก** และมีค่า $112 - 81 = 31$

### ขั้น 2 — บวกทีละหลัก

**ทำอะไร:** บวกแถบบิต**เดิม**ทั้งสองตรง ๆ **ทำไม:** ห้ามเอา $-81$ ที่แปลงได้ไปใช้แทนนะครับ — ที่แปลงเมื่อกี้ทำเพื่อ**อ่านค่า**เท่านั้น เวลาบวกจริงต้องใช้แถบ 2's complement เดิมคือ $10101111$

**ข้อ 2b · 01110000 + 10101111**

```
     1 1
   0 1 1 1 0 0 0 0
+  1 0 1 0 1 1 1 1
  -----------------
1  0 0 0 1 1 1 1 1
^ ตัวทดออกนอกกล่อง 8 บิต ทิ้งไป
```

### ขั้น 3 — ทิ้งตัวทดตัวสุดท้าย

ผลบวกดิบได้ **9 บิต** คือ $1\,00011111$ ครับ แต่กล่องเรามีแค่ 8 บิต
ตามกฎ `"discard any final carry bit"` → **ทิ้งเลข 1 ตัวหน้าสุดไป** เหลือ `00011111`

**ทำไมทิ้งได้โดยคำตอบยังถูก**

กล่อง 8 บิตครบรอบทุก $2^8 = 256$ ครับ
ตัวทดที่ล้นออกมามีน้ำหนัก 256 พอดี — และในระบบ 2's complement เลข $-81$ ถูกเก็บเป็น $256 - 81 = 175$
เพราะฉะนั้น $112 + 175 = 287 = 256 + 31$ → พอทิ้ง 256 (ตัวทด) ก็เหลือ **31** ซึ่งคือคำตอบจริงเป๊ะครับ

### ขั้น 4 — อ่านค่าฐานสิบ

คำตอบ $00011111$ — บิตซ้ายสุด = 0 → เป็นบวก อ่านตรง ๆ

| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|
| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| 0 | 0 | 0 | 16 | 8 | 4 | 2 | 1 |

$16 + 8 + 4 + 2 + 1 = \mathbf{31}$ ครับ

### ★ ขั้น 5 — ตรวจ overflow (หัวใจของข้อนี้)

บิตเครื่องหมายของตัวตั้ง = **0** (บวก) และ **1** (ลบ) → **ต่างกัน**
→ ตามที่หัวข้อ 6 พิสูจน์ไว้ กรณีเครื่องหมายต่างกัน **ไม่มีทางเกิด overflow** เลยครับ เพราะผลลัพธ์ยิ่งเข้าใกล้ศูนย์ ไม่มีทางล้นกล่อง

ต่อให้อยากเช็คซ้ำก็ได้: ผลลัพธ์ $00011111$ บิตเครื่องหมาย = 0 (บวก) ซึ่งตรงกับตัวตั้งที่ใหญ่กว่า ($+112$) → สมเหตุสมผล ✓

**★ กับดักตัวจริงของข้อ 2b**

**ข้อนี้มีตัวทดออกนอกกล่อง แต่ไม่ overflow ครับ**
คนที่จำผิดว่า "มีตัวทดล้น = overflow" จะตอบว่า invalid ทันที — **ผิดครับ**
สไลด์เขียนไว้ชัดว่า overflow ดูจาก **incorrect sign bit** เท่านั้น และกฎ "discard any final carry bit" ก็บอกอยู่แล้วว่าตัวทดตัวนั้น**ต้องทิ้ง** ไม่ได้แปลว่ามีปัญหา
ข้อนี้แหละครับที่อาจารย์ตั้งใจใส่มาเพื่อวัดว่าน้องแยกสองเรื่องนี้ออกไหม

**④ คำตอบข้อ 2b**

**01110000 + 10101111 = 00011111** (ทิ้งตัวทดตัวที่ 9 แล้ว)
ค่าฐานสิบ: $(+\mathbf{112}) + (-\mathbf{81}) = +\mathbf{31}_{10}$
**ไม่มี overflow (valid)** — เพราะเครื่องหมายของตัวตั้งต่างกัน

**⑤ ตรวจคำตอบด้วยวิธีที่สอง**

**วิธีที่ 1 — ฐานสิบ:** $112 + (-81) = 31$ ✓ ตรงกับ $00011111_2 = 31$
**วิธีที่ 2 — มองเป็นการลบ:** $(+112) + (-81)$ ก็คือ $112 - 81 = 31$ ✓
**วิธีที่ 3 — เช็คด้วยพิสัย:** 31 อยู่ในช่วง $-128 \cdots + 127$ สบาย ๆ → ยืนยันว่าไม่ overflow ✓
**วิธีที่ 4 — เช็คตัวทดเข้า/ออกบิตเครื่องหมาย:** ตัวทดที่**เข้า**บิตซ้ายสุด $= 1$ และตัวทดที่**ออก**จากบิตซ้ายสุด $= 1$ → **เท่ากัน = ไม่ overflow** ✓ (ตรงกับผลที่ได้จากการดูเครื่องหมายเป๊ะ)

**หัวใจของข้อ 2**

**ข้อ 2a วัดว่าบวกเป็นไหม · ข้อ 2b วัดว่าน้องแยก "ตัวทดล้นออกนอกกล่อง" ออกจาก "overflow" ได้ไหม — ซึ่งเป็นคนละเรื่องกันโดยสิ้นเชิงครับ**

# 11 เฉลย Assignment 2 · ข้อ 3 (ลบ)

**ข้อ 3** Assignment #2

> Perform each subtraction in the 2's complement form:
> a. 00110011 − 00010000
> b. 01100101 − 11101000

แปล: หาผลลบในระบบ 2's complement ทั้งสองข้อครับ

**① โจทย์ซ่อนอะไรไว้ (ใช้กับทั้ง a และ b)**

**ซ่อนไว้ 4 อย่างครับ**

① คำว่า `subtraction in the 2's complement form` แปลว่า **ห้ามลบตรง ๆ** ต้องแปลงเป็นการบวกก่อน — ถ้าน้องยืมทีละหลักแบบเลขฐานสิบ ถึงจะได้คำตอบถูก ก็**ผิดโจทย์** เพราะโจทย์สั่งวิธีมาแล้วครับ

② ต้องทำ 2's complement เฉพาะ**ตัวหลัง** (subtrahend) เท่านั้น

③ ข้อ b ตัวลบ 11101000 ขึ้นต้นด้วย **1** → เป็นเลข**ลบ** → โจทย์คือ “ลบด้วยเลขลบ” = **บวก** ผลลัพธ์จะ**ใหญ่ขึ้น** ไม่ใช่เล็กลง — ตรงนี้คือกับดักตัวจริง

④ ทั้งสองข้อต้องเช็ค overflow เพราะข้อ b จะกลายเป็น “บวก + บวก” ซึ่งเป็นกรณีเสี่ยงครับ

**② เตรียมก่อนลงมือ (ใช้กับทั้ง a และ b)**

**ใช้วิธีไหน:** 3 ขั้นตายตัวตามสไลด์หน้า 11 — `take the 2's complement of the subtrahend and add. Discard any final carry bit.`

**ทำไมเลือกวิธีนี้:** เพราะคอมพิวเตอร์ไม่มีวงจรลบครับ มีแต่วงจรบวก การลบจึงถูกแปลงเป็น “บวกกับตัวตรงข้าม” เสมอ — และโจทย์ก็สั่งมาตรง ๆ ว่าให้ใช้วิธีนี้

**กฎที่จะใช้:** · $A - B = A + (2\text{'s complement of } B)$ · ทิ้งตัวทดตัวสุดท้าย · overflow ดูจากเครื่องหมายของ**ตัวตั้งที่ใช้บวกจริง** (คือ $A$ กับ 2's complement ของ $B$)

## ข้อ 3a — 00110011 − 00010000

### ③ ไล่ทีละขั้น · ขั้น 1 — ระบุ minuend / subtrahend และอ่านค่า

**ทำอะไร:** แยกให้ชัดว่าตัวไหนตั้ง ตัวไหนลบ **ทำไม:** ถ้าทำ 2's complement ผิดตัว คำตอบจะกลับเครื่องหมายทั้งข้อครับ

**minuend (ตัวตั้ง)** 00110011 — **บิตซ้ายสุด 0 → บวก**

| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|
| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| 0 | 0 | 32 | 16 | 0 | 0 | 2 | 1 |

**subtrahend (ตัวลบ)** 00010000 — **บิตซ้ายสุด 0 → บวก**

| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|
| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| 0 | 0 | 0 | 16 | 0 | 0 | 0 | 0 |

$00110011_2 = 32 + 16 + 2 + 1 = +\mathbf{51}$ · $00010000_2 = +\mathbf{16}$

โจทย์คือ $51 - 16$ → คาดว่าได้ $+\mathbf{35}$ ครับ (จดไว้ไปตรวจตอนท้าย)

### ขั้น 2 — ทำ 2's complement ของตัวลบ

**ทำอะไร:** แปลง $+16$ ให้เป็น $-16$ **ทำไม:** เพื่อเปลี่ยนโจทย์ลบให้เป็นโจทย์บวกครับ

ข้อ 3a · ทำ 2's complement ของตัวลบ 00010000

```
   0 0 0 1 0 0 0 0   ตัวลบเดิม
   1 1 1 0 1 1 1 1   กลับทุกบิต (1's complement)
+  0 0 0 0 0 0 0 1   บวก 1
   ---------------
   1 1 1 1 0 0 0 0   = 2's complement
```

ได้ `11110000` ครับ

**ตรวจว่าถูกไหม:** $11110000 \to 00001111 \to +1 \to 00010000 = 16$ ✓ เพราะฉะนั้น $11110000$ คือ $-16$ จริง

### ขั้น 3 — บวก minuend เข้ากับ 2's complement ที่ได้

ข้อ 3a · 00110011 + 11110000 (2's comp ของ 00010000)

```
    1 1 1
   0 0 1 1 0 0 1 1
+  1 1 1 1 0 0 0 0
---------------------
1  0 0 1 0 0 0 1 1
^ ตัวทดออกนอกกล่อง 8 บิต ทิ้งไป
```

### ขั้น 4 — ทิ้งตัวทดตัวสุดท้าย แล้วอ่านค่า

ผลบวกดิบได้ 9 บิต $1\,00100011$ → ทิ้งเลข 1 ตัวหน้า → เหลือ `00100011`

คำตอบ $00100011$ — บิตซ้ายสุด = 0 → เป็นบวก อ่านตรง ๆ

| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|
| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| 0 | 0 | 32 | 0 | 0 | 0 | 2 | 1 |

$32 + 2 + 1 = \mathbf{35}$ ครับ

### ขั้น 5 — ตรวจ overflow

การบวกจริงคือ $(+51) + (-16)$ → บิตเครื่องหมายของตัวตั้ง = **0** และ **1** → **ต่างกัน**
→ **ไม่มีทางเกิด overflow** ครับ (ผลลัพธ์เข้าใกล้ศูนย์เสมอ)

**④ คำตอบข้อ 3a**

**00110011 − 00010000 = 00110011 + 11110000 = 00100011**

ค่าฐานสิบ: $(+\mathbf{51}) - (+\mathbf{16}) = +\mathbf{35}_{10}$ · **ไม่มี overflow (valid)**

**⑤ ตรวจคำตอบด้วยวิธีที่สอง**

**วิธีที่ 1 — ฐานสิบ:** $51 - 16 = 35$ ✓ ตรงกับ $00100011_2 = 35$

**วิธีที่ 2 — บวกกลับ:** คำตอบ + ตัวลบ ต้องได้ตัวตั้งคืน → $00100011 + 00010000 = 00110011$ ✓ ได้ $35 + 16 = 51$ กลับมาเป๊ะ

**วิธีที่ 3 — เช็คเครื่องหมายที่คาดไว้:** ตัวตั้ง $(51)$ ใหญ่กว่าตัวลบ $(16)$ → คำตอบต้องเป็น**บวก** → บิตซ้ายสุดต้องเป็น 0 ✓

**⑥ กับดักของข้อ 3a**

① **ลืมทิ้งตัวทด** ถ้าตอบเป็น 100100011 (9 บิต) คือผิดครับ — กล่องมี 8 บิต ต้องทิ้งเสมอ

② **ทำ 2's complement ผิดตัว** ถ้าเผลอทำของ 00110011 แทน จะได้ $11001101 + 00010000 = 11011101 = -35$ → **เครื่องหมายกลับด้าน** ครับ

③ **ลบตรง ๆ แบบยืมหลัก** ถึงจะได้ 35 เหมือนกัน แต่ผิดโจทย์ เพราะโจทย์สั่งว่า **"in the 2's complement form"** — ต้องโชว์ขั้นทำ 2's complement ให้เห็นครับ

## ข้อ 3b — 01100101 − 11101000 (ลบด้วยเลขลบ — ต้องเช็ค overflow)

### ③ ไล่ทีละขั้น · ขั้น 1 — ระบุ minuend / subtrahend และอ่านค่า

**minuend (ตัวตั้ง)** 01100101 — **บิตซ้ายสุด 0 → บวก**

| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|
| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| 0 | 64 | 32 | 0 | 0 | 4 | 0 | 1 |

$01100101_2 = 64 + 32 + 4 + 1 = +\mathbf{101}$

ส่วน subtrahend 11101000 บิตซ้ายสุดเป็น **1** → เป็น**เลขลบ** หาขนาดด้วยการทำ 2's complement กลับ: 11101000 → 00010111 → $+1$ → 00011000 $= 24$ → เพราะฉะนั้น $11101000_2 = -\mathbf{24}$

**★ อ่านโจทย์ให้ออกก่อนลงมือ**

โจทย์จริง ๆ คือ $(+\mathbf{101}) - (-\mathbf{24})$ ครับ

**ลบด้วยเลขลบ = บวก** → คำตอบต้องเป็น $101 + 24 = \mathbf{125}$ **ซึ่งใหญ่ขึ้น**

ใครที่เผลออ่าน 11101000 เป็น 232 แล้วคิดว่า $101 - 232$ ติดลบ — จะตอบผิดตั้งแต่ยังไม่เริ่มคำนวณครับ

และ 125 นี้**ใกล้เพดาน** $+127$ **มาก** → ยิ่งต้องเช็ค overflow ให้ดี

### ขั้น 2 — ทำ 2's complement ของตัวลบ

**ทำอะไร:** แปลง $-24$ ให้เป็น $+24$ **ทำไม:** กฎเดิมเป๊ะครับ — 2's complement คือการ**กลับเครื่องหมาย** ไม่ว่าตัวเดิมจะเป็นบวกหรือลบก็ใช้ได้ทั้งคู่

**ข้อ 3b · ทำ 2's complement ของตัวลบ 11101000**

```
   1 1 1 0 1 0 0 0   ตัวลบเดิม
   0 0 0 1 0 1 1 1   กลับทุกบิต (1's complement)
+  0 0 0 0 0 0 0 1   บวก 1
-------------------
   0 0 0 1 1 0 0 0   = 2's complement
```

ได้ **00011000** $= 16 + 8 = +\mathbf{24}$ ครับ

**ตรวจว่าถูกไหม:** 00011000 → 11100111 → $+1$ → 11101000 ✓ กลับมาเป็นตัวเดิมได้ → แสดงว่าเป็นคู่ตรงข้ามกันจริง

### ขั้น 3 — บวก

ตอนนี้โจทย์กลายเป็น $(+101) + (+24)$ ครับ — **สังเกตให้ดี: ตอนนี้เป็นบวกทั้งคู่แล้ว = กรณีเสี่ยง overflow**

ข้อ 3b · 01100101 + 00011000 (2's comp ของ 11101000)

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

### ขั้น 4 — อ่านค่าฐานสิบ

**ไม่มี**ตัวทดออกนอกกล่องเลย (carry out $= 0$) ผลลัพธ์คือ `01111101`

คำตอบ 01111101 — บิตซ้ายสุด = 0 → เป็นบวก อ่านตรง ๆ

| 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|
| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| 0 | 64 | 32 | 16 | 8 | 4 | 0 | 1 |

$64 + 32 + 16 + 8 + 4 + 1 = \mathbf{125}$ ครับ

### ★ ขั้น 5 — ตรวจ overflow (หัวใจของข้อนี้)

การบวกจริงคือ $(+101) + (+24)$ → บิตเครื่องหมายของตัวตั้ง = **0** และ **0** → **เหมือนกัน = กรณีเสี่ยง ต้องเช็คต่อ**
บิตเครื่องหมายของผลลัพธ์ 01111101 = **0** (บวก) → **เหมือนกับตัวตั้ง = เครื่องหมายไม่พลิก = ไม่เกิด overflow** ครับ

**ยืนยันด้วยพิสัย:** $125 \leq 127$ → ยัง**อยู่ในกล่อง** 8 บิตพอดี ✓
เฉียดมากครับ — ถ้าตัวตั้งเป็น $+104$ แทน ($104 + 24 = 128 > 127$) จะ overflow ทันที

**④ คำตอบข้อ 3b**

**01100101 − 11101000 = 01100101 + 00011000 = 01111101**
ค่าฐานสิบ: $(+\mathbf{101}) - (-\mathbf{24}) = +\mathbf{125}_{10}$ · **ไม่มี overflow (valid)**

**⑤ ตรวจคำตอบด้วยวิธีที่สอง**

**วิธีที่ 1 — ฐานสิบ:** $101 - (-24) = 101 + 24 = 125$ ✓ ตรงกับ $01111101_2 = 125$
**วิธีที่ 2 — บวกกลับ:** คำตอบ + ตัวลบเดิม ต้องได้ตัวตั้งคืน → $01111101 + 11101000 = 1\,01100101$ → ทิ้งตัวทด → 01100101 ✓ ได้ $125 + (-24) = 101$ กลับมาเป๊ะ
**วิธีที่ 3 — เช็คพิสัย:** 125 อยู่ในช่วง $-128 \cdots + 127$ → ยืนยันว่าไม่ overflow ✓
**วิธีที่ 4 — ตัวทดเข้า/ออกบิตเครื่องหมาย:** ตัวทดที่**เข้า**บิตซ้ายสุด $= 0$ · ตัวทดที่**ออก**จากบิตซ้ายสุด $= 0$ → เท่ากัน → ไม่ overflow ✓

**⑥ กับดักของข้อ 3b — ข้อนี้มีกับดักเยอะที่สุด**

① **อ่านตัวลบเป็นเลขไม่มีเครื่องหมาย** ถ้าอ่าน 11101000 เป็น 232 น้องจะคิดว่าโจทย์คือ $101 - 232 = -131$ ซึ่งนอกจากผิดแล้วยัง overflow อีกต่างหาก — ผิดสองชั้นครับ

② **เผลอ "บวก" แทนที่จะทำ 2's complement ก่อน** ถ้าเอา $01100101 + 11101000$ ตรง ๆ จะได้ $01001101 = 77$ ซึ่งคือ $101 + (-24)$ = คำตอบของ**โจทย์บวก** ไม่ใช่โจทย์ลบครับ

③ **คิดว่า "ลบแล้วต้องเล็กลง"** ข้อนี้ตัวลบเป็นเลขลบ → ผลลัพธ์**ใหญ่ขึ้น** ($101 \rightarrow 125$) ซึ่งถูกต้องแล้วครับ ไม่ต้องตกใจ

④ **ข้ามการเช็ค overflow เพราะเห็นว่า "ไม่มีตัวทดล้น"** — ข้อนี้ไม่มีตัวทดล้นก็จริง แต่หลังแปลงแล้วเป็น **บวก + บวก** ซึ่งเป็นกรณีเสี่ยง **ต้องเช็คทุกครั้ง** ครับ (บังเอิญข้อนี้รอดมาได้เพราะ $125 \leq 127$ พอดี)

**หัวใจของข้อ 3**

**ข้อ 3a วัดว่าน้องจำขั้นตอน "2's complement แล้วบวก" ได้ไหม · ข้อ 3b วัดว่าน้องอ่านเครื่องหมายของตัวลบออกไหม และรู้ไหมว่า "ลบด้วยเลขลบ" เปลี่ยนกรณีจากปลอดภัยเป็นกรณีเสี่ยง overflow ครับ**

## สรุปคำตอบ Assignment 2 ข้อ 1–3 ในหน้าเดียว

| ข้อ | โจทย์ | คำตอบ | ฐานสิบ / หมายเหตุ |
|---|---|---|---|
| 1 | $-99.03125_{10}$ → single precision | **1 10000101 10001100001000000000000 = $C2C61000_{16}$** | sign $1 \cdot e = 6 \cdot$ biased $= 133$ |
| 2a | 00010110 + 00110011 | **01001001** | $+22 + 51 = +73 \cdot$ no overflow |
| 2b | 01110000 + 10101111 | **00011111** (ทิ้งตัวทด) | $+112 + (-81) = +31 \cdot$ **มีตัวทดล้น แต่ไม่ overflow** |
| 3a | 00110011 − 00010000 | **00100011** (ทิ้งตัวทด) | $+51 - 16 = +35 \cdot$ no overflow |
| 3b | 01100101 − 11101000 | **01111101** | $+101 - (-24) = +125 \cdot$ no overflow (เฉียด $+127$) |

## 12 กับดักรวมของ EP นี้

ก่อนไปหน้าสรุป พี่ขอรวบกับดักทั้งหมดของ EP03 ไว้ที่เดียวครับ ถ้าน้องเช็คครบ 8 ข้อนี้ก่อนส่งกระดาษ โอกาสพลาดจะเหลือน้อยมาก

| # | กับดัก | คำตอบผิดที่จะได้ | วิธีกัน |
|---|---|---|---|
| 1 | **เก็บบิต 1 หน้าจุดลง mantissa** | mantissa ขึ้นต้นด้วย 1 · เลขฐานสิบหกเพี้ยนทั้งชุด | ท่องว่า **hidden bit ไม่เก็บ** — เก็บเฉพาะสิ่งที่อยู่**หลังจุด** |
| 2 | **นับ mantissa ไม่ครบ 23 บิต** | แถบรวมได้ 31 หรือ 33 บิต | นับเป็นกลุ่มละ 5: $5{+}5{+}5{+}5{+}3 = 23$ |
| 3 | **ลืมบวก bias 127** | $e = 6$ กลายเป็น 00000110 แทน 10000101 | เขียน “$+127$” กำกับไว้ทุกครั้งที่หา exponent |
| 4 | **ถอดรหัสแล้วยัง “บวก” 127** | $e$ ผิดไป 254 → ค่ามหาศาล | เข้ารหัส **บวก** · ถอดรหัส **ลบ** — ตรงข้ามกันเสมอ |
| 5 | **เอา 2's complement ไปใช้กับช่อง exponent** | $e = -3$ กลายเป็น 11111101 แทน 01111100 | ช่อง exponent ใช้ **bias** เท่านั้น ไม่มี 2's complement |
| 6 | **★ คิดว่าตัวทดล้น = overflow** | ตอบ invalid ทั้งที่คำตอบถูก (เช่นข้อ 2b) | ดูที่ **บิตเครื่องหมาย** เท่านั้น — ตัวทดล้นให้ทิ้ง |
| 7 | **อ่านแถบที่ขึ้นต้นด้วย 1 เป็นเลขบวก** | 11101000 อ่านเป็น 232 แทน $-24$ | เห็นบิตซ้ายสุดเป็น 1 → **ทำ 2's complement กลับก่อนเสมอ** |
| 8 | **ทำ 2's complement ผิดตัวตอนลบ** | ได้คำตอบที่เครื่องหมายกลับด้าน | ทำเฉพาะ **subtrahend** (ตัวหลังเครื่องหมายลบ) เท่านั้น |

**กับดักข้อที่ 6 คือข้อที่ออกสอบแน่นอนที่สุด**

พี่ขอย้ำอีกครั้งเป็นประโยคเดียวครับ:

**“ตัวทดที่ล้นออกนอกกล่อง 8 บิต ให้ทิ้งเสมอ และมันไม่ได้แปลว่าคำตอบผิด — overflow ดูจากบิตเครื่องหมายที่พลิกเท่านั้น”**

ในเฉลย Assignment ข้อ 2b น้องเห็นแล้วว่า**มี**ตัวทดล้นแต่คำตอบถูก และในตัวอย่างสไลด์หน้า 9 น้องเห็นว่า**ไม่มี**ตัวทดล้นแต่ overflow เต็ม ๆ ครับ

## 13 สรุป EP03 หน้าเดียว

### EP03 · Floating Point + เลขคณิตจำนวนมีเครื่องหมาย — ทั้งหมดอยู่ในหน้านี้ครับ

**กล่อง single precision · 32 บิต**

**sign 1 + exponent 8 + mantissa 23**

bias = **127** (double: 11 บิต, bias 1023)

สูตร bias = $2^{n-1} - 1$ เมื่อ $n$ = จำนวนบิต exponent

**บวกแบบ 2's complement**

**บวกตรง ๆ แล้วทิ้งตัวทดตัวสุดท้าย**

ไม่ต้องแยกกรณีเครื่องหมายเลย วงจรเดียวจบ

คำตอบขึ้นต้น 1 → ทำ 2's comp กลับก่อนอ่านค่า

**สูตร 6 ขั้น (ฐานสิบ → 32 บิต)**

① sign: บวก = **0** · ลบ = **1**

② แปลงฐานสอง: หาร 2 / คูณ 2

③ normalize เป็น **1.m × 2ᵉ**

④ exponent = **e + 127** → 8 บิต

⑤ mantissa = เศษหลังจุด เติม 0 ครบ **23**

⑥ ต่อกัน 32 บิต → จับ 4 บิต → ฐาน 16

**normalize · hidden bit**

ส่วนจำนวนเต็ม $k$ หลัก → **e = k − 1**

เลขน้อยกว่า 1 → เลื่อนขวา → **e ติดลบ**

บิต 1 หน้าจุด **ไม่เก็บ** (รู้อยู่แล้วว่าเป็น 1)

**ถอดรหัสย้อนกลับ**

① ซอย 1 / 8 / 23

② sign: 0 = บวก · 1 = ลบ

③ $e$ = **exponent − 127** (ลบ ไม่ใช่บวก)

④ เติม “1.” หน้า mantissa แล้วเลื่อนจุด $e$ ที่

**เช็คเร็วก่อนส่ง**

ขนาด > 1 → exponent ขึ้นต้น **1**

ขนาด < 1 → exponent ขึ้นต้น **0111**

นับ 23 · นับ 8 · รวมได้ 32

**★ overflow — กฎเดียวที่ต้องจำ**

**เครื่องหมายตัวตั้งเหมือนกัน แต่ผลลัพธ์พลิก = overflow**

เกิดได้แค่ **(+)+(+)** และ **(−)+(−)**

เครื่องหมายต่างกัน → **ไม่มีทาง** overflow

**ตัวทดล้น ≠ overflow** — คนละเรื่องกัน

อีกวิธี: **carry เข้า ≠ carry ออก ที่บิตเครื่องหมาย**

**ลบ = บวกกับ 2's complement**

$A - B = A +$ **2's comp ของ B**

ทำเฉพาะ **subtrahend** (ตัวหลัง) เท่านั้น

ทางลัด: คัดลอกจากขวาจนเจอ 1 ตัวแรก แล้วกลับที่เหลือ

**คูณ / หาร — ต่างจากบวกลบ**

ต้องแปลงเป็น **true form** ก่อนเสมอ

คูณ = partial products เลื่อนซ้ายทีละช่อง

หาร = **ลบซ้ำแล้วนับรอบ** (รอบที่ติดลบไม่นับ)

เครื่องหมาย: เหมือนกัน → บวก · ต่างกัน → ลบ (ทำ 2's comp)

**พิสัย 8 บิต 2's complement**

**−128 ถึง +127**

$10000000 = -128$ (ทำ 2's comp แล้วได้ตัวเดิม)

$01111111 = +127$

## เช็คลิสต์ก่อนสอบ — ทำได้ทุกข้อไหมครับ

| ✓ | ทำได้ไหมครับ |
|---|---|
| ☐ | บอกได้ว่ากล่อง single precision แบ่งเป็นกี่ช่วง ช่วงละกี่บิต และ bias เท่าไหร่ |
| ☐ | อธิบายได้ว่า**ทำไม**ต้องบวก bias 127 (ไม่ใช่แค่ท่องว่าบวก) |
| ☐ | อธิบายได้ว่า**ทำไม**บิต 1 หน้าจุดไม่ต้องเก็บ |
| ☐ | แปลงเลขฐานสิบที่มีเศษ เป็น 32 บิต และเป็นฐานสิบหก ได้ครบ 6 ขั้น |
| ☐ | แปลงเลขที่**น้อยกว่า 1** ได้ (exponent ติดลบ) เช่น $-0.15625$ |
| ☐ | อ่านย้อนกลับจาก 32 บิต เป็นเลขฐานสิบได้ (อย่าลืม**ลบ** 127) |
| ☐ | บวกเลขมีเครื่องหมายได้ครบทั้ง 4 กรณี และอ่านค่าฐานสิบของคำตอบได้ |
| ☐ | บอกได้ว่า overflow เกิดเมื่อไหร่ และ**ทำไมตัวทดล้นไม่ใช่ overflow** |
| ☐ | ลบด้วยวิธี 2's complement ได้ และรู้ว่าต้องทำ complement ตัวไหน |
| ☐ | คูณด้วย partial products ได้ และใส่เครื่องหมายให้ผลคูณถูก |
| ☐ | หารด้วยการลบซ้ำได้ และรู้ว่ารอบที่ทำให้ติดลบ**ไม่นับ** |

| ✓ | ทำได้ไหมครับ |
|---|---|
| ☐ | ทำ Assignment 2 ข้อ 1–3 ได้เองโดยไม่เปิดเฉลย |

**EP ต่อไปเจออะไร**

ถึงตรงนี้น้องเก็บตัวเลขได้ครบทุกแบบแล้วครับ — จำนวนเต็มไม่มีเครื่องหมาย (EP01), จำนวนเต็มมีเครื่องหมาย (EP02), และเลขทศนิยม (EP03) รวมถึงบวกลบคูณหารได้หมดแล้ว

**แต่พี่ขอถามต่อครับ — ถ้าข้อมูลวิ่งผ่านสายแล้วมีบิตพลิกกลางทางล่ะครับ เราจะรู้ได้ยังไงว่าข้อมูลเสีย**
สมมติน้องส่ง $01000001$ ไป แต่ปลายทางได้รับ $01000011$ (บิตเดียวพลิก) — ปลายทางไม่มีทางรู้เลยว่าอันไหนคือของจริง เพราะทั้งสองแถบก็เป็นเลขที่ถูกต้องทั้งคู่
EP04 จะสอนวิธี**ใส่ยามเฝ้าให้ข้อมูล** ครับ — **parity bit** (ยามคนเดียว บอกได้แค่ว่ามีบิตเสีย), **CRC** (ยามที่จับได้แม้เสียหลายบิต) และ **Hamming code** (ยามที่**ชี้ได้เลยว่าบิตไหนเสีย** แล้วซ่อมได้ด้วย) พร้อมกับรหัสอีก 3 ตระกูลคือ **BCD**, **Gray code** และ **ASCII** ครับ

## แหล่งอ้างอิงของ EP นี้

| แหล่ง | หน้า / ข้อ | เนื้อหาที่ใช้ |
|---|---|---|
| **EES 271-02 Operation and codes-2026 Rev4.pdf** | หน้า 2 | โครงสร้าง floating point · single/double precision · 3 ส่วนของกล่อง |
| **— เล่มเดียวกัน** | หน้า 3 | ขั้นตอนการแปลง 4 ขั้น · normalized form · bias 127 / 1023 |
| **— เล่มเดียวกัน** | หน้า 4 | Example: Convert $-118.625_{10}$ (สไลด์เว้นว่างไว้ — ชีทนี้เติมเต็มให้) |
| **— เล่มเดียวกัน** | หน้า 6–7 | กฎการบวกเลขมีเครื่องหมาย 4 กรณี · Example (a)–(d) |
| **— เล่มเดียวกัน** | หน้า 9 | Overflow condition · Example $01111101 + 00111010$ |
| **— เล่มเดียวกัน** | หน้า 11 | การลบ · Example $-25 - 19$ |
| **— เล่มเดียวกัน** | หน้า 13–14 | การคูณ partial products · Example $0110 \times 1100$ |
| **— เล่มเดียวกัน** | หน้า 16–18 | การหารด้วยการลบซ้ำ · Example $01100100 \div 00011001$ และ $00110010 \div 00001111$ |
| **EES271 Assignment #2** | ข้อ 1, 2a–b, 3a–b | เฉลยเต็มอยู่ในหัวข้อ 9–11 ของชีทนี้ |
| **Floyd, Digital Fundamentals, 10th ed.** | บทที่ 2 | Signed numbers · Arithmetic operations with signed numbers · Floating-point numbers |

**ปิดท้ายจากพี่**

EP นี้ไม่มีอะไรยากเลยครับ มีแต่**ขั้นตอนเยอะ** — และคะแนนที่หายไปส่วนใหญ่ไม่ได้หายเพราะคิดไม่ออก แต่หายเพราะ**นับบิตพลาด** กับ**ลืมเช็ค overflow**
เพราะฉะนั้นเวลาซ้อม ให้ซ้อม**ขั้นตอนให้เป็นนิสัย** ไม่ต้องรีบครับ ทำช้า ๆ แต่ครบทุกขั้น ดีกว่าทำเร็วแล้วต้องกลับมาไล่ใหม่ทั้งข้อครับ