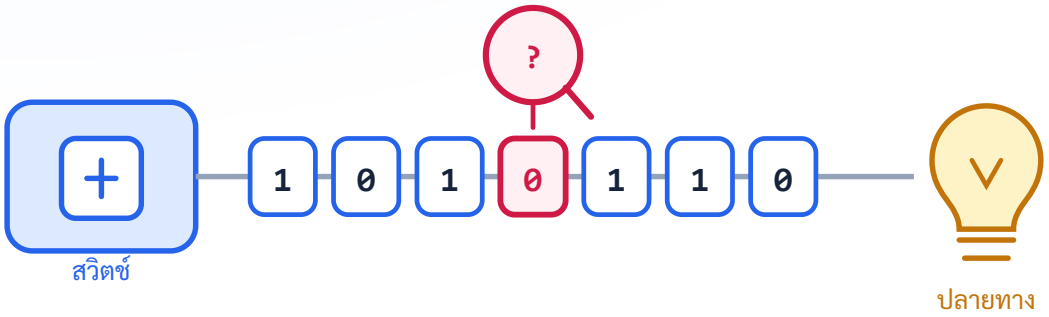
04 EP

รหัสและการตรวจจับข้อผิดพลาด
ส่งบิตไปแล้ว รู้ได้ยังไงว่าไม่เพี้ยน

BCD · Gray code · ASCII · parity คู่/คี่ · CRC กับการหาร XOR · Hamming code ที่บอกได้ว่าผิดตรงไหน



บิตหนึ่งตัวพลิกกลางทาง — ปลายทางจะรู้ได้ยังไงครับ

พี่สอนน้องแบบไม่ต้องมีพื้นฐานมาก่อนเลยครับ · อ้างอิงสไลด์ EES271 Week 2 (หน้า 14–35) + Assignment #2 ข้อ 4–10

ตอนนี้เราจะเดินไปทางไหนกัน

ก่อนเริ่ม พี่ขอกางแผนที่ให้น้องดูก่อนครับ EP นี้แบ่งเป็นสองครึ่งชัดเจน — **ครึ่งแรกคือ “รหัสสำหรับแทนของ”** (หัวข้อ 2–4) และ **ครึ่งหลังคือ “รหัสสำหรับกันข้อมูลพัง”** (หัวข้อ 5–7) แล้วปิดท้ายด้วยเฉลย Assignment ข้อ 4–10 ครบทุกข้อย่อยครับ

1	**ทำไมต้องมี “รหัส” ทั้งที่เรามีเลขฐานสองอยู่แล้ว**	ภาพรวม
2	**BCD — เขียนเลขฐานสิบทีละหลัก และ 6 รหัสต้องห้าม**	ออกสอบชัวร์
3	**Gray code — เปลี่ยนทีละ 1 บิตเสมอ + แปลงไป-กลับ**	จุดที่คนสับสน
4	**ASCII — เขียนตัวอักษรด้วย 0 กับ 1**	อ่านผ่าน ๆ ได้
5	**Parity — บิตเดียวที่บอกว่าข้อมูลเพี้ยนหรือเปล่า**	พื้นฐานของ 6 กับ 7
6	**CRC — เอาข้อมูลไปหาร แล้วส่งเศษไปด้วย**	ต้องฝึกมือ
7	**Hamming code — บอกได้ด้วยว่าผิดตรงไหน**	ยากที่สุด
8	**เฉลย Assignment ข้อ 4–6 (BCD และ Gray code)**	เฉลยเต็ม
9	**เฉลย Assignment ข้อ 7–8 (Parity)**	เฉลยเต็ม
10	**เฉลย Assignment ข้อ 9 (CRC) — ทุกรอบการหาร**	ข้อยาก
11	**เฉลย Assignment ข้อ 10 (Hamming) — ทุกกลุ่มพาริตี**	ข้อยากที่สุด
12	**กับดักรวม + สรุปหน้าเดียว + เช็คลิสต์ก่อนสอบ**	ทวนก่อนสอบ

วิธีใช้ชีทนี้ — I DO / WE DO / YOU DO

ทุกหัวข้อสอนของ EP นี้จบด้วยลำดับเดียวกันสามขั้นครับ อย่าข้ามขั้นไหนนะครับ

I DO — พี่ทำให้ดู
พี่ทำโจทย์ให้ดูทีละขั้น **ทุกขั้นบอกด้วยว่าทำไมถึงทำแบบนั้น** ไม่ใช่แค่บอกว่าทำอะไร ครับ น้องแค่อ่านตามให้เข้าใจ

WE DO — ทำด้วยกัน
พี่เว้นช่องว่างไว้ให้น้องเติม **ลองเติมเองก่อนแล้วค่อยดูเฉลยข้างล่าง** เฉลยจะอธิบายทุกช่องพร้อมเหตุผลครับ

YOU DO — ตาน้อง
น้องทำเองทั้งข้อ **เฉลยอยู่ติดกันทันที** แต่พี่ขอให้ทำให้จบก่อนค่อยเลื่อนดูนะครับ ไม่งั้นจะได้แค่ความรู้สึกว่า “ง่ายจัง”

ข้อตกลงเรื่องภาษา

โจทย์ทุกข้อพี่พิมพ์เป็นภาษาอังกฤษตามข้อสอบจริง แล้วอธิบายวิธีคิดเป็นภาษาไทยข้างล่างครับ เพราะข้อสอบ EES271 เป็นภาษาอังกฤษล้วน น้องต้องคุ้นกับคำว่า `assign the proper even parity bit` · `determine the transmitted CRC` · `encode in Hamming code using even parity` ให้ได้ครับ

และทุกเฉลยจะปิดท้ายด้วย**ประโยคภาษาอังกฤษที่ลอกลงกระดาษข้อสอบได้เลย** เพื่อให้น้องเขียนตอบเป็นอังกฤษได้ด้วย ไม่ใช่แค่คิดออกอย่างเดียว

คะแนนของวิชานี้

Assignment 25% · Midterm 35% · Final 40% ครับ — **เฉลย Assignment ในหัวข้อ 8 ถึง 11 คือส่วนที่กินคะแนน 25% โดยตรง** และข้อ 9 กับข้อ 10 คือสองข้อที่ยากที่สุดของ Assignment ชุดนี้ พี่จึงทำละเอียดที่สุดครับ

1 ทำไมต้องมี “รหัส” ทั้งที่เรามีเลขฐานสองอยู่แล้ว

ตะขอเกี่ยว · รับจาก EP03

จบ EP03 ตอนนี้เก็บตัวเลขได้ครบทุกแบบแล้วครับ — เลขเต็มก็ได้ เลขติดลบก็ได้ด้วย 2's complement เลขทศนิยมก็ได้ด้วย IEEE single precision 32 บิต

แต่ถ้าข้อมูลวิ่งผ่านสายแล้วมีบิตพลิกกลางทางล่ะครับ เราจะรู้ได้ยังไงว่าข้อมูลเสีย — ปลายทางได้เลข 32 บิตมาก้อนหนึ่ง มันหน้าตาเหมือนเลขปกติทุกประการ ไม่มีอะไรบอกเลยว่า “อันนี้เพี้ยนนะ” ครับ EP นี้คือคำตอบของคำถามนั้น

ของใกล้ตัว: ส่งกระดาษโน้ตข้ามโต๊ะ

สมมติน้องเขียนโน้ตว่า **“ยืมยางลบ 3 ก้อน”** แล้วส่งต่อ ๆ กันไปห้าโต๊ะกว่าจะถึงปลายทางครับ ระหว่างทางมีคนทำหมึกหยดใส่ เลข 3 กลายเป็นเลข 8 — **ปลายทางอ่านได้ชัดเจนว่า “8 ก้อน” และไม่มีทางรู้เลยว่ามันเคยเป็น 3**

ปัญหาคือกระดาษแผ่นนั้น**ไม่ได้พกอะไรมาให้ตรวจสอบตัวเองเลย**ครับ ถ้าน้องเขียนเพิ่มไปอีกบรรทัดว่า “ตัวเลขในโน้ตนี้เป็นเลขคี่” ปลายทางก็จับได้ทันทีว่ามีอะไรผิด — **นั่นคือแนวคิดทั้งหมดของ error detection ครับ: ส่งข้อมูลไปพร้อมกับ “ของแถม” ที่เอาไว้ตรวจกันเอง**



ฝั่งรับเห็นแค่ 0 กับ 1 — ไม่มีอะไรบอกว่าบิตไหนเพี้ยน

ทางแก้: ส่ง “บิตตรวจสอบ” แนบไปด้วย แล้วให้ฝั่งรับคำนวณซ้ำเทียบกัน

ทั้ง EP นี้อยู่บนภาพเดียวนี้ครับ — ข้อมูลวิ่งบนสาย มีโอกาสพลิก และเราต้องจับให้ได้

EP นี้มีรหัส 2 ครอบครัว อย่าจำปนกัน

คำว่า “code” ใน EP นี้โผล่มา 5 ตัว แต่มันทำคนละหน้าที่กันครับ พี่แยกให้ชัดตั้งแต่ต้นเลย จะได้ไม่งงตอนอ่านโจทย์

ครอบครัวที่ 1 — รหัสสำหรับ “แทนของ”

เอาไว้**เขียนของอย่างหนึ่งให้เป็น 0 กับ 1** ครับ ไม่ได้ตรวจอะไรทั้งสิ้น

BCD — แทนเลขฐานสิบทีละหลัก
Gray code — แทนลำดับที่เปลี่ยนทีละ 1 บิต
ASCII — แทนตัวอักษร

ครอบครัวที่ 2 — รหัสสำหรับ “กันพัง”

เอาไว้**ตรวจว่าข้อมูลเสียหรือเปล่า** ครับ ต้องมีบิตส่วนเกินแถมไปด้วยเสมอ

Parity — บิตเดียว ตรวจได้ แก้ไม่ได้
CRC — เศษจากการหาร ตรวจได้เก่งกว่า แก้ไม่ได้
Hamming code — ตรวจได้ และ**บอกตำแหน่ง**ที่ผิดด้วย

ตารางที่ต้องเห็นภาพก่อนเริ่ม: ใครทำอะไรได้บ้าง

วิธี	บิตส่วนเกิน	รู้ว่าผิด	รู้ว่าผิดตรงไหน	จับ error ได้แค่ไหน
Parity	1 บิต	ได้	ไม่ได้	จับได้เฉพาะบิตพลิก**จำนวนคี่** (1, 3, 5 บิต) — พลิก 2 บิตหลุด
CRC	r บิต (r = ความยาว G ลบ 1)	ได้	ไม่ได้	จับได้ทั้งบิตเดี่ยว บิตคู่ และ burst error ยาว ๆ
Hamming	p บิต ($2^p \geq d + p + 1$)	ได้	ได้	แก้ได้จริงเฉพาะ**บิตเดียว** ถ้าพลิก 2 บิตจะแก้ผิดที่

อ่านตารางนี้ยังไง

สังเกตแนวโน้มครับ — **ยิ่งอยากรู้ละเอียดขึ้น ต้องจ่ายบิตส่วนเกินมากขึ้น** parity จ่าย 1 บิตได้แค่ “ผิด/ไม่ผิด” ส่วน Hamming จ่าย 4 บิตถึงจะได้ “ผิดที่ตำแหน่งไหน” ไม่มีของฟรีในวิชานี้ครับ ข้อสอบชอบถามเปรียบเทียบตรงนี้

เทียบของเก่า · EP01–EP03

EP01 ถึง EP03 ทั้งหมดเราสนใจแค่ **“ค่า”** ของเลขครับ — 1101 มีค่าเท่าไร บวกลบยังไง แต่ EP นี้เราเปลี่ยนมุมมองไปสนใจ **“รูปแบบการเขียน”** แทน — บิตชุดเดียวกันเป๊ะ ๆ ถ้าตกลงกันคนละแบบ ก็แปลออกมาคนละความหมาย เพราะฉะนั้น**โจทย์จะบอกเสมอว่านี่คือรหัสอะไร** (เขียนห้อยไว้ว่า $_{\text{BCD}}$ หรือ $_{\text{Gray}}$ หรือ $_{2}$) — **อ่านตัวห้อยให้ดีก่อนลงมือทุกครั้งครับ**

2 BCD — เขียนเลขฐานสิบทีละหลัก

ตะขอเกี่ยว

EP01 น้องแปลง 2469_{10} เป็นฐานสองด้วยการหารสั้นเก็บเศษได้แล้วใช่ไหมครับ ได้เลขยาว ๆ ออกมาก้อนหนึ่ง

สมมติว่าถ้าน้องต้องทำหน้าจอนาฬิกาดิจิทัลที่โชว์เลข 2469 ละครับ — หน้าจอมีช่องแยกกัน 4 ช่อง แต่ละช่องโชว์ได้ทีละหลัก น้องจะเอาเลขฐานสองก้อนเดียวยาว ๆ นั้น ไปบอกช่องแต่ละช่องได้ยังไงว่า “เธอโชว์เลข 2 นะ” “เธอโชว์เลข 4 นะ” ครับ

ของใกล้ตัว: หน้าจอเครื่องคิดเลขกับนาฬิกาดิจิทัล

ลองมองหน้าจอเครื่องคิดเลขใกล้ ๆ ครับ มันไม่ใช่จอเดียวรวด — มัน**คือหลาย ๆ ช่องเรียงกัน แต่ละช่องมีขีดไฟ 7 ขีดของตัวเอง** และแต่ละช่องรู้จักแค่เลข 0 ถึง 9 เท่านั้น ไม่รู้จักเลข 2469 ทั้งก้อนเลย

เพราะฉะนั้นวงจรที่ป้อนให้หน้าจอ**ต้องส่งข้อมูลไปทีละหลัก**ครับ และแต่ละหลักต้องเป็น 0 กับ 1 เพราะวงจรพูดได้แค่นั้น — นี่แหละคือ BCD



2469 ในรูป BCD คือ 0010 0100 0110 1001 — สี่ก้อนแยกกัน ไม่ใช่เลขก้อนเดียว

DEFINITION · Binary-Coded Decimal

Binary-coded decimal (BCD) is a code in which each decimal digit is represented by its own group of four bits, using only the codes 0000_2 through 1001_2.

พี่แปลว่า: **BCD คือการเขียนเลขฐานสิบ “ทีละหลัก” โดยแต่ละหลักได้ 4 บิตเป็นของตัวเอง และใช้ได้แค่รหัส 0000 ถึง 1001 เท่านั้น**ครับ ห้ามข้ามหลักไปยุ่งกัน

ที่มา: ทำไมต้อง 4 บิตพอดี ไม่ใช่ 3 ไม่ใช่ 5

ตรงนี้พี่ไม่ให้ท่องครับ คิดเอาได้เลย — เราต้องเขียนเลข **0 ถึง 9 คือ 10 ตัว** ให้ได้ครบใช่ไหมครับ แล้วบิต n ตัวเขียนรหัสได้ 2^n แบบ (อันนี้คือ 2^n ตัวเดิมจาก EP01 เลยครับ)

3 บิต	4 บิต	5 บิต
$2^3 = 8$ แบบ	$2^4 = 16$ แบบ	$2^5 = 32$ แบบ
น้อยกว่า 10 — ไม่พอครับ	มากกว่า 10 — พอครับ	พอ แต่เปลืองเกินจำเป็น

4 บิตคือจำนวนที่**น้อยที่สุดที่ยังพอ**ครับ เลยเลือกอันนี้

เลขฐานสิบ	รหัส BCD	เลขฐานสิบ	รหัส BCD
0	0000	5	0101
1	0001	6	0110
2	0010	7	0111
3	0011	8	1000
4	0100	9	1001

ทางลัด · ไม่ต้องท่องตารางนี้

ตารางนี้**ไม่มีอะไรใหม่เลยครับ** — มันคือเลขฐานสอง 4 บิตธรรมดาของ 0 ถึง 9 เป๊ะ ๆ ที่น้องแปลงเป็นมาตั้งแต่ EP01 แล้ว

เพราะฉะนั้นเวลาสอบ**ไม่ต้องจำ** แค่เขียนน้ำหนัก **8 4 2 1** ไว้ข้างบน แล้วเลือกให้บวกได้เท่าตัวเลขที่ต้องการ เช่น 6 = 4 + 2 ⟹ 0110 ครับ

6 รหัสต้องห้าม — 1010 ถึง 1111

4 บิตทำรหัสได้ 16 แบบ แต่เราใช้แค่ 10 แบบ เพราะฉะนั้น**เหลือรหัสที่ไม่มีเจ้าของอยู่ 6 ตัว**ครับ คือ 1010, 1011, 1100, 1101, 1110, 1111 (ค่าฐานสิบคือ 10 ถึง 15)

ในโลกของเลขฐานสองธรรมดา 6 ตัวนี้ใช้ได้ปกติ — 1111 ก็คือ 15 ครับ แต่**ในโลกของ BCD ทั้ง 6 ตัวนี้ผิดกฎทันที** เพราะเลขฐานสิบหนึ่งหลัก**ไม่มีทางเป็น 10 ถึง 15 ได้** หลักหนึ่งหลักมีค่าได้แค่ 0 ถึง 9 เท่านั้นครับ

รหัส 4 บิตทั้ง 16 แบบ — ใช้ได้ 10 ห้ามใช้ 6

0000 = 0	0001 = 1	0010 = 2	0011 = 3	0100 = 4	0101 = 5	0110 = 6	0111 = 7
1000 = 8	1001 = 9	1010 ห้าม	1011 ห้าม	1100 ห้าม	1101 ห้าม	1110 ห้าม	1111 ห้าม

เจอ 6 รหัสแดงในสตริง BCD เมื่อไหร่ = ข้อมูลนั้นไม่ใช่ BCD ที่ถูกต้อง (invalid code)

ข้อสอบชอบให้สตริงมาแล้วถามว่า “เป็น BCD ที่ถูกต้องไหม” — วิธีตอบคือหั่น 4 บิตแล้วไล่ดูว่าเจอตัวแดงหรือเปล่าครับ

กับดักใหญ่ที่สุดของ BCD — BCD ไม่ใช่เลขฐานสองปกติ

ตรงนี้พี่ขอเน้นมาก ๆ ครับ เพราะเป็นจุดที่ทำคะแนนหลุดมากที่สุดของหัวข้อนี้ ลองดูตารางเทียบข้างล่างนี้ให้เห็นกับตาก่อนเลย — **เลขตัวเดียวกันเป๊ะ ๆ แต่เขียนออกมาคนละเรื่องเลย**

เลขฐานสิบ	BCD (หลักละ 4 บิต)	เลขฐานสองปกติ (base 2)	เหมือนกันไหม
9	1001	1001	**เหมือน** — เพราะมีหลักเดียวและไม่เกิน 9 ครับ กรณีนี้เท่านั้นที่บังเอิญตรงกัน
47	0100 0111	101111	**ไม่เหมือน** — BCD ใช้ 8 บิต ส่น base 2 ใช้ 6 บิต และเรียงบิตคนละแบบ
250	0010 0101 0000	11111010	**ไม่เหมือน** — BCD ยาว 12 บิต base 2 ยาว 8 บิต
8013	1000 0000 0001 0011	1111101001101	**ไม่เหมือน** — ยิ่งเลขยาว BCD ยิ่งเปลืองบิตกว่า

กับดัก · อันนี้ห้ามพลาดเด็ดขาด

1. อย่าเอา BCD ไปคิดค่าเป็นเลขฐานสองก้อนเดียว — เช่น $0100\,0111_{\text{BCD}}$ ถ้าน้องหลงไปคิดว่า $01000111_2 = 71_{10}$ ก็ผิดทันทีครับ คำตอบที่ถูกคือ **47** เพราะต้องอ่านทีละก้อน 4 บิต: 0100 = 4 และ 0111 = 7

2. เจอตัวห้อยให้อ่านให้ดี — $_{\text{BCD}}$ กับ $_2$ ต่างกันคนละโลกครับ ข้อสอบเขียนตัวห้อยเล็ก ๆ ไว้เสมอ ถ้าอ่านข้ามคือทำผิดทั้งข้อ

เลขที่มีจุดทศนิยมก็ทำเหมือนกันเป๊ะ

จุดทศนิยม**ไม่ได้เปลี่ยนกฎอะไรเลยครับ** — หลักคือหลัก แต่ละหลักได้ 4 บิตเหมือนเดิม แค่ต้อง**คงจุดไว้ตรงที่เดิม**เท่านั้นเอง

17.1 ⟹ BCD (ตัวอย่างจากสไลด์)

```
   1        7     .     1
0001     0111     .  0001
─────────────────────────
ตอบ  0001 0111 . 0001
```

ทางลัด · แปลงกลับ

ขาแปลงกลับ (BCD ⟹ ฐานสิบ) ให้**ยืนที่จุดทศนิยมแล้วหั่น 4 บิตออกไปสองทาง**ครับ — ทางซ้ายหั่นไปทางซ้าย ทางขวาหั่นไปทางขวา ห้ามหั่นจากปลายสุด

ถ้าไม่มีจุดทศนิยม ให้หั่น**จากขวาสุด**เข้ามาครับ แล้วถ้าก้อนซ้ายสุดเหลือไม่ครบ 4 บิต ให้เติม 0 นำหน้าได้เลย

ศัพท์สอบของหัวข้อนี้

English	แปลว่า	ข้อสอบเขียนมาแบบนี้
binary-coded decimal (BCD)	รหัสฐานสิบเข้ารหัสฐานสอง	Convert the following numbers using binary-coded decimals
decimal digit	หลักของเลขฐานสิบ	Each decimal digit is represented by 4 bits
invalid code	รหัสที่ใช้ไม่ได้	1010 through 1111 are invalid BCD codes
inefficient for storage	เปลืองที่เก็บ	BCD is inefficient for storage

I DO พี่ทำให้ดูก่อนครับ

ตัวอย่างที่พี่ทำ — โจทย์พี่แต่งเอง · ปูทางก่อนไป Assignment

> Convert 605_{10} to BCD.

แปล: จงแปลง 605 ฐานสิบ ให้เป็นรหัส BCD ครับ

1 — 6 0 5

หั่นเลขออกเป็นหลัก ๆ ก่อน

ทำแบบนี้ก่อนเสมอครับ **เพราะ BCD ทำงานทีละหลัก** ไม่ได้มองเลขทั้งก้อน — ได้ 3 หลักคือ 6, 0, 5

2 — 8 4 2 1 / 0 1 1 0 / 6 = 4+2 ; 8 4 2 1 / 0 1 0 1 / 5 = 4+1

แปลงแต่ละหลักด้วยน้ำหนัก 8-4-2-1

เพราะ 4 บิตของ BCD มีน้ำหนักประจำหลักเหมือนเลขฐานสองปกติ

ครับ: 6 = 4+2 ⟹ 0110 · 0 ⟹ 0000 · 5 = 4+1 ⟹ 0101

3 — เรียงตามลำดับหลักเดิม: 0110 0000 0101

ต่อกลับเข้าไปตามลำดับเดิม

ลำดับห้ามสลับครับ เพราะหลักร้อยต้องอยู่ซ้ายสุดเสมอ — ตอบ:

$605_{10} = 0110\,0000\,0101_{\text{BCD}}$

4 — อ่านกลับทีละ 4 บิต: 0110 0000 0101 = 605

ตรวจด้วยวิธีที่สอง — แปลงกลับ

เพราะการตรวจด้วยวิธีเดิมจะพลาดซ้ำรอยเดิมครับ ต้องเดินย้อนทาง:

0110 = 6 · 0000 = 0 · 0101 = 5 ⟹ ได้ 605 ตรงกับโจทย์ ✓

WE DO ทำด้วยกันครับ พี่เว้นช่องไว้

ทำด้วยกัน โจทย์พี่แต่งเอง

> Convert 1984_{10} to BCD, then convert $1001\,0110\,0011_{\text{BCD}}$ back to decimal.

ข้อนี้ทำสองทางครับ ไปทางหนึ่งแล้วกลับอีกทางหนึ่ง

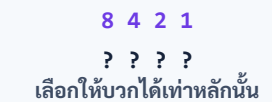
1



หั่นเป็นหลัก

1984 มี ..... หลัก ⟹ ผลลัพธ์ BCD ต้องยาว ..... บิต

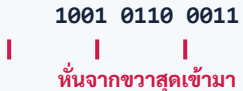
2



แปลงทีละหลัก

1 ⟹ ..... 9 ⟹ ..... 8 ⟹ ..... 4 ⟹ .....

3



ขากลับ — หั่น 4 บิต

1001 = ... 0110 = ... 0011 = ... ⟹ ตอบ .....

เฉลย WE DO — เติมทุกช่องพร้อมเหตุผล

ขั้น 1 — 1984 มี 4 หลัก ⟹ ผลลัพธ์ยาว 16 บิต เพราะ BCD ให้หลักละ 4 บิตพอดี ไม่มีการรวบหรือย่อ เพราะฉะนั้น $4 \times 4 = 16$ บิตครับ จำนวนบิตที่คำนวณไว้ล่วงหน้าแบบนี้เอาไว้**ตรวจว่าเขียนตกหล่นไหม**ตอนจบ

ขั้น 2 — แปลงทีละหลักด้วยน้ำหนัก 8-4-2-1 ทำไมใช้ 8-4-2-1 ได้ เพราะ 4 บิตของ BCD ก็คือเลขฐานสอง 4 บิตธรรมดา น้ำหนักประจำหลักจึงเป็น $2^3\ 2^2\ 2^1\ 2^0$ ครับ:

1 = 1 ⟹ **0001** · **9** = 8+1 ⟹ **1001** · **8** = 8 ⟹ **1000** · **4** = 4 ⟹ **0100**

ขั้น 3 — ต่อเรียงตามลำดับหลักเดิม เพราะถ้าสลับที่ ค่าจะเปลี่ยนทันที (0001 1001 ไม่เท่ากับ 1001 0001): $1984_{10} = 0001\,1001\,1000\,0100_{\text{BCD}}$ — นับได้ 16 บิตพอดีตามที่ทำนายไว้ ✓

ขั้น 4 — ขากลับ หั่น 4 บิตจากขวาสุด เพราะโจทย์ไม่มีจุดทศนิยม จุดอ้างอิงจึงเป็นปลายขวา (ถ้าหั่นจากซ้ายแล้วก้อนสุดท้ายเหลือไม่ครบ 4 บิตจะเพี้ยนทั้งแถว): 1001 | 0110 | 0011

ขั้น 5 — อ่านทีละก้อน 1001 = 8+1 = **9** · 0110 = 4+2 = **6** · 0011 = 2+1 = **3**

ขั้น 6 — ตรวจด้วยวิธีที่สอง เอา 963 แปลงกลับเป็น BCD: 9 ⟹ 1001, 6 ⟹ 0110, 3 ⟹ 0011 ได้ 1001 0110 0011 ตรงกับโจทย์ ✓ และไม่มีก้อนไหนตกอยู่ในช่วงต้องห้าม 1010–1111 เลย

ตอบ: $1984_{10} = 0001\,1001\,1000\,0100_{\text{BCD}}$ **และ** $1001\,0110\,0011_{\text{BCD}} = 963_{10}$

“Each decimal digit is encoded separately in its own four bits, so 1984 becomes 0001 1001 1000 0100.”

YOU DO ตาน้องครับ ทำเสร็จแล้วค่อยดูเฉลยข้างล่าง

Y2.1

> Convert $0111\,0100_{\text{BCD}}$ to a decimal number. Then state what value the same bit string would have if it were read as an ordinary base-2 number.

ข้อนี้พี่จงใจให้เห็นว่าอ่านคนละแบบได้คนละคำตอบครับ

เฉลย Y2.1

ขั้น 1 — ดูตัวห้อยก่อน เพราะตัวห้อยบอกว่าต้องอ่านด้วยกฎไหน โจทย์เขียน $_{\text{BCD}}$ เพราะฉะนั้นต้อง**หั่นทีละ 4 บิต** ครับ: 0111 | 0100

ขั้น 2 — ตรวจก่อนว่าทุกก้อนถูกกฎไหม เพราะถ้ามีก้อนไหนตกช่วง 1010–1111 แปลว่าโจทย์ให้สตริงที่ไม่ใช่ BCD มา: 0111 = 7 (ผ่าน) · 0100 = 4 (ผ่าน) ✓

ขั้น 3 — อ่านทีละก้อนด้วยน้ำหนัก 8-4-2-1 0111 = 4+2+1 = **7** และ 0100 = 4 = **4** เรียงตามเดิมได้ **74**

ขั้น 4 — ถ้าอ่านเป็นฐานสองปกติแทน คราวนี้มองเป็นเลขก้อนเดียว 8 บิต น้ำหนักคือ 128-64-32-16-8-4-2-1: $01110100_2 = 64 + 32 + 16 + 4 = \mathbf{116}$

ตอบ: อ่านแบบ BCD ได้ 74 · อ่านแบบฐานสองปกติได้ 116 — ต่างกันคนละเรื่องเลยครับ

“Read as BCD the string is 74; read as a plain binary number the same string is 116.”

Y2.2

> Convert 39.5_{10} to BCD.

มีจุดทศนิยมด้วยครับ ระวังตำแหน่งจุด

เฉลย Y2.2

ขั้น 1 — หั่นเป็นหลัก โดยถือว่าจุดทศนิยมเป็นเส้นแบ่ง เพราะ BCD สนใจ “หลัก” ไม่ได้สนใจค่ารวม: ฝั่งซ้ายจุดมี 3 กับ 9 ฝั่งขวาจุดมี 5

ขั้น 2 — แปลงทีละหลักเหมือนเดิม ทำไมยังใช้ 8-4-2-1 ได้ทั้งที่มันเป็นทศนิยม เพราะ**ตัวเลขในหลักยังเป็น 0 ถึง 9 เหมือนเดิม**ครับ ความเป็นทศนิยมอยู่ที่ตำแหน่งจุด ไม่ได้อยู่ที่ค่าของหลัก:

3 = 2+1 ⟹ **0011** · 9 = 8+1 ⟹ **1001** · 5 = 4+1 ⟹ **0101**

ขั้น 3 — วางจุดกลับที่เดิม จุดอยู่หลังหลักหน่วย (คือหลัง 9) ในผลลัพธ์จึงต้องอยู่หลังก้อน 1001

ขั้น 4 — ตรวจด้วยวิธีที่สอง อ่านย้อนกลับ: 0011 = 3, 1001 = 9, จุด, 0101 = 5 ⟹ 39.5 ตรงกับโจทย์ ✓

ตอบ: $39.5_{10} = 0011\,1001\,.\,0101_{\text{BCD}}$

“39.5 in BCD is 0011 1001 . 0101, keeping the decimal point in the same place.”

Y2.3

> Is $1011\,0010$ a valid BCD code? Explain your answer.

ข้อแบบนี้ออกสอบบ่อยครับ ตอบสั้น ๆ แต่ต้องมีเหตุผล

เฉลย Y2.3

ขั้น 1 — หั่น 4 บิต เพราะกฎของ BCD บังคับให้แต่ละหลักยาว 4 บิตพอดี: 1011 | 0010

ขั้น 2 — ตรวจทีละก้อนว่าตกอยู่ในช่วง 0000–1001 ไหม เพราะนี่คือช่วงเดียวที่ BCD ยอมรับ:

ก้อนแรก **1011** มีค่าเท่ากับ 11 ฐานสิบ ซึ่ง**เกิน 9** ⟹ ตกอยู่ในกลุ่ม 6 รหัสต้องห้าม

ก้อนที่สอง 0010 = 2 ⟹ ผ่านปกติ

ขั้น 3 — สรุป ขอแค่มีก้อนเดียวที่ผิดกฎ ทั้งสตริงก็ใช้ไม่ได้แล้วครับ เพราะเลขฐานสิบหนึ่งหลัก**ไม่มีทางมีค่า 11** ได้

ตอบ: ไม่ใช่ BCD ที่ถูกต้อง — เพราะกลุ่ม 1011 มีค่า 11 ซึ่งเกิน 9

“No. The group 1011 equals 11, which is one of the six invalid BCD codes (1010 to 1111).”

3 Gray code — รหัสที่เปลี่ยนทีละ 1 บิตเสมอ

ตะขอเกี่ยว

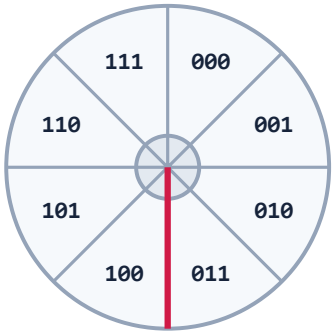
หัวข้อที่แล้วน้องเห็นแล้วว่าเรามีสิทธิ์เลือกได้ว่าจะ “เขียน” เลขด้วยรหัสแบบไหน — BCD เลือกเขียนทีละหลักเพื่อให้ต่อหน้าจองง่าย

สมมติว่าถ้าน้องต้องอ่านค่ามุมจากจานที่กำลังหมุนอยู่ ละครับ — ตอนจานหมุนผ่านรอยต่อ เซนเซอร์อาจอ่านค่าเก่ากับค่าใหม่ปนกันได้เสี้ยววินาที ถ้าเลขฐานสองปกติเปลี่ยนทีเดียว 4 บิต น้องคิดว่าเซนเซอร์จะอ่านออกมาเป็นอะไรครับ

ของใกล้ตัว: จานหมุนวัดมุม (shaft encoder)

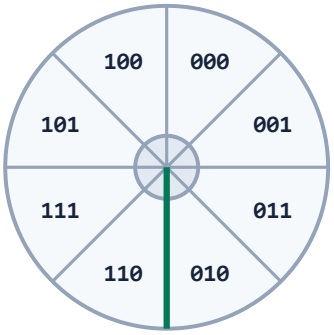
จานหมุนวัดมุมคือแผ่นกลม ๆ ติดอยู่กับแกนหมุน บนแผ่นแบ่งเป็นช่อง ๆ แต่ละช่องเจาะรูไว้ไม่เหมือนกัน มีลำแสงยิงผ่านลงมา **รูทะลุ = 1 · ทึบ = 0** ครับ เซนเซอร์อ่านหลายแถวพร้อมกันได้เป็นรหัสหนึ่งชุด เอาไปแปลว่าตอนนี้แกนหันไปทางไหน

ปัญหาอยู่ตรงนี้ครับ: **เซนเซอร์แต่ละตัวไม่ได้เรียงตรงกันเป๊ะ 100%** ตอนจานผ่านรอยต่อพอดี ตัวหนึ่งอาจเห็นค่าใหม่แล้ว แต่อีกตัวยังเห็นค่าเก่าอยู่



(a) เลขฐานสองปกติ
รอยต่อแดง 011 ⟶ 100 เปลี่ยน 3 บิตพร้อมกัน

vs



(b) Gray code
รอยต่อเขียว 010 ⟶ 110 เปลี่ยนแค่ 1 บิต

จานเดียวกัน หมุนเหมือนกัน ต่างกันแค่ “เลือกเขียนรหัสยังไง” เท่านั้นเองครับ

เกิดอะไรขึ้นถ้าเซนเซอร์เหลื่อมกันนิดเดียว

ฐานสองปกติ: 011 กำลังจะกลายเป็น 100

`0 1 1 → 1 0 0`

ระหว่างทางอาจอ่านได้ 111 · 000 · 001 · 110

อ่านผิดเป็นตำแหน่งที่อยู่คนละฟากของจานได้เลยครับ

Gray: 010 กำลังจะกลายเป็น 110

`0 1 0 → 1 1 0`

ระหว่างทางอ่านได้แค่ 010 หรือ 110 เท่านั้น

แย่ที่สุดคือคลาดไปแค่ช่องเดียว ไม่มีทางกระโดดข้ามฟาก

เพราะมีบิตเดียวที่กำลังเปลี่ยน ค่าที่อ่านได้จึงเป็นได้แค่ “ค่าเก่า” หรือ “ค่าใหม่” — ไม่มีค่าที่สาม

DEFINITION · Gray Code

The Gray code is unweighted and is not an arithmetic code; there are no specific weights assigned to the bit positions. Its advantage is that it exhibits only a single bit change from one code word to the next in sequence.

พี่แปลว่า: **Gray code ไม่มีน้ำหนักประจำหลัก เอาไปบวกลบไม่ได้ หน้าที่เดียวของมันคือ ทำให้รหัสที่อยู่ติดกันในลำดับ ต่างกันแค่ 1 บิตเสมอ**ครับ

กับดัก · unweighted แปลว่าอะไร

คำว่า **unweighted** สำคัญมากครับ — แปลว่า **ห้ามเอาน้ำหนัก 8-4-2-1 มาคูณ** Gray code 1110 **ไม่ได้** มีค่า $8+4+2=14$ นะครับ ถ้าอยากรู้ค่าจริงต้อง**แปลงกลับเป็นฐานสองก่อนเสมอ** ($1110_{\text{Gray}} = 1011_2 = 11$) — คนละเรื่องกันเลยครับ

ตารางเทียบ 0 ถึง 15 — ดูคอลัมน์ “กี่บิตที่เปลี่ยน”

ตารางนี้คือหัวใจของทั้งหัวข้อครับ พี่เพิ่มคอลัมน์ **“เทียบกับแถวบน เปลี่ยนไปกี่บิต”** ให้ดู เพราะนั่นคือสิ่งเดียวที่ทำให้ Gray ต่างจากฐานสองปกติ

Decimal	Binary	บิตที่เปลี่ยน (binary)	Gray Code	บิตที่เปลี่ยน (Gray)	หมายเหตุ
0	0000	—	0000	—	เริ่มเหมือนกัน
1	0001	1	0001	1	
2	0010	2	0011	1	
3	0011	1	0010	1	
4	0100	3	0110	1	

Decimal	Binary	บิตที่เปลี่ยน (binary)	Gray Code	บิตที่เปลี่ยน (Gray)	หมายเหตุ
5	0101	1	0111	**1**	
6	0110	**2**	0101	**1**	
7	0111	1	0100	**1**	
8	**1000**	**4 บิตพร้อมกัน**	**1100**	**1**	จุดที่แย่ที่สุด
9	1001	1	1101	**1**	
10	1010	**2**	1111	**1**	
11	1011	1	1110	**1**	
12	1100	**3**	1010	**1**	
13	1101	1	1011	**1**	
14	1110	**2**	1001	**1**	
15	1111	1	1000	**1**	

สังเกตให้ดี 3 อย่าง

1. คอลัมน์ Gray เป็นเลข **1** ทั้งแถวตั้งแต่ต้นจนจบ ไม่มีข้อยกเว้นเลยครับ

2. คอลัมน์ binary กระโดดได้ถึง **4 บิต** ตอน $0111 \rightarrow 1000$ — ตรงนั้นคือจุดที่จานหมุนอ่านพลาดง่ายที่สุด

3. 0, 1 ทั้งสองรหัสเหมือนกัน แต่ตั้งแต่ 2 ขึ้นไปเริ่มแตกทางแล้ว — **อย่าจำผิดว่า Gray = binary** ครับ

วิธีที่ 1: binary ⟹ Gray (บิตแรกลอกมา แล้ว XOR คู่ที่ติดกัน)

ก่อนอื่นทวนก่อนครับว่า **XOR** (modulo-2) คืออะไร — มันคือ **“ต่างกันได้ 1 เหมือนกันได้ 0”** เท่านั้นเองครับ ในสไลด์เรียกว่า “add and discard carries” คือบวกแล้วทิ้งตัวทด ซึ่งได้ผลเหมือนกันเป๊ะ

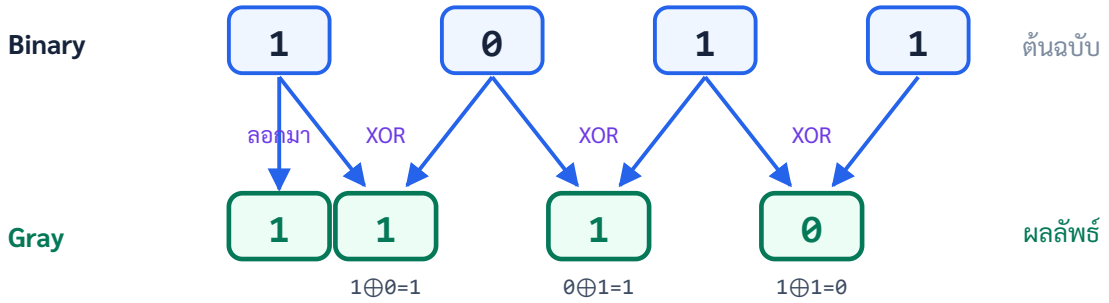
a	b	$a \oplus b$	อ่านว่า
0	0	0	เหมือนกัน
0	1	1	ต่างกัน
1	0	1	ต่างกัน
1	1	0	เหมือนกัน

RULE · Binary-to-Gray

1. The most significant bit (left-most) in the Gray code is the same as the corresponding MSB in the binary number.

2. Going from left to right, add each adjacent pair of binary code bits to get the next Gray code bit. Discard carries.

พี่แปลสั้น ๆ: **ตัวซ้ายสุดลอกมาเลย จากนั้น XOR ทีละคู่ที่ติดกัน ของ "เลขฐานสองต้นฉบับ" ทั้งหมด** — สังเกตว่า**ทุกคู่หยิบจากแถวบนอย่างเดียว**ครับ



$1011_2 \Rightarrow 1110_{\text{Gray}}$ — ลูกศรทุกคู่**ออกจากแถวบนเท่านั้น** ไม่มีเส้นไหนวิ่งกลับขึ้นไปเลยครับ

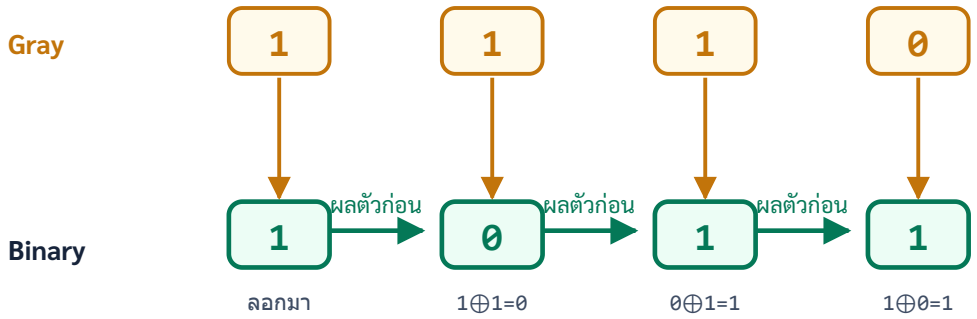
วิธีที่ 2: Gray ⟹ binary (ทิศต่างจากขาไป ต้องดูให้ดี)

RULE · Gray-to-Binary

1. The most significant bit (left-most) in the binary code is the same as the corresponding bit in the Gray code.

2. Add each binary code bit generated to the Gray code bit in the next adjacent position. Discard carries.

ตรงนี้คือจุดที่คนสับสนมากที่สุดครับ พี่ขอเน้น: **ขาไป** เรา XOR **"บิตต้นฉบับกับบิตต้นฉบับ"** แต่ **ขากลับ** เรา XOR **"ผลลัพธ์ที่เพิ่งได้ กับ บิต Gray ตัวถัดไป"** — แปลว่า**ขากลับต้องเดินทีละก้าว เพราะต้องรอผลลัพธ์ตัวก่อนหน้าเสมอ** ข้ามไปทำตัวขวาสุดก่อนไม่ได้ครับ



$1110_{\text{Gray}} \Rightarrow 1011_2$ — ลูกศรเขียวคือ “ผลลัพธ์ที่เพิ่งได้” วิ่งไปทบกับ Gray ตัวถัดไป **ทบต่อกันไปเรื่อย ๆ** ไม่เหมือนขาไปที่ทุกคู่หยิบจากแถวบนล้วน

ทางลัด · ตรวจงานฟรี ๆ ทุกครั้ง

แปลงเสร็จแล้ว**แปลงกลับทันที**ครับ ถ้าได้ต้นฉบับคืนแปลว่าถูก — ใช้เวลาเพิ่มไม่ถึงครึ่งนาที แต่กันพลาดได้ทั้งข้อ

อีกอันหนึ่ง: **จำนวนบิตต้องเท่ากันเสมอ** เข้าไปกี่บิต ออกมาต้องกี่บิต ถ้าได้ไม่เท่าเดิมแปลว่าลืมทำบางตำแหน่งครับ (โจทย์ให้ 8 บิต ต้องตอบ 8 บิต ห้ามตัด 0 นำหน้าทิ้ง)

กับดัก · สับสนสองทิศทาง

อาการที่เจอบ่อยที่สุดคือ**ทำขากลับด้วยสูตรขาไป** ครับ — คือไป XOR คู่ที่ติดกันของ Gray เอง ซึ่งจะได้คำตอบผิดทันทีตั้งแต่บิตที่สาม

วิธีกันพลาด: **ดูว่าโจทย์ให้อะไรมา** — ถ้าให้ $_2$ มาแล้วขอ Gray คือขาไป (คู่จากแถวบนล้วน) ถ้าให้ $_{\text{Gray}}$ มาแล้วขอ base 2 คือขากลับ (ทบต่อกัน) ครับ

ศัพท์สอบของหัวข้อนี้

English	แปลว่า	ข้อสอบเขียนมาแบบนี้
Gray code	รหัสเกรย์	Convert the following binary number to Gray code
unweighted code	รหัสที่ไม่มีน้ำหนักประจำหลัก	The Gray code is unweighted
single bit change	เปลี่ยนทีละบิตเดียว	only a single bit change from one code word to the next
adjacent pair	คู่ที่อยู่ติดกัน	add each adjacent pair of binary code bits
discard carries	ทิ้งตัวทด (คือ XOR นั่นเอง)	Discard carries
shaft encoder	ตัวเข้ารหัสมุมแกนหมุน	Why Gray code on a shaft encoder?

I DO พี่ทำให้ดูทั้งสองทิศครับ

ตัวอย่างที่พี่ทำ — โจทย์พี่แต่งเอง (สไลด์หน้า 27 ให้ 8 บิต ไว้ทำใน WE DO)

> Convert 110010_2 to Gray code, then convert the result back to base 2 to check.

แปล: แปลง 110010 ฐานสอง เป็น Gray แล้วแปลงกลับมาตรวจครับ

1

1 1 0 0 1 0

1 _ _ _ _ _

MSB ลอกลงมาตรง ๆ

ลอกบิตซ้ายสุดลงมา

เพราะกฎข้อ 1 บอกว่า MSB ของ Gray เท่ากับ MSB ของ binary เสมอครับ ⟹ Gray เริ่มด้วย **1**

2

1 1 0 0 1 0

1⊕1 1⊕0 0⊕0 0⊕1 1⊕0

0 1 0 1 1

XOR ทุกคู่ที่ติดกัน จากซ้ายไปขวา

เพราะกฎข้อ 2 บอกว่า Gray บิตถัดไป = ผลบวกของคู่ที่ติดกันโดยทิ้งตัวทดครับ มีทั้งหมด 5 คู่ (6 บิตให้ 5 คู่): 1⊕1=0, 1⊕0=1, 0⊕0=0, 0⊕1=1, 1⊕0=1

3

1 0 1 0 1 1

ต่อ MSB เข้ากับ 5 บิตที่ได้

ตอบ: $110010_2 = 101011_{\text{Gray}}$ — นับได้ 6 บิตเท่าต้นฉบับพอดี ✓

4

1 0 1 0 1 1

ทบต่อกันไปทางขวา

1 1 0 0 1 0

ตรวจด้วยวิธีที่สอง — แปลงกลับ

เพราะต้องเดินย้อนทางถึงจะจับผิดตัวเองได้ครับ: ลอก 1 มา · 1⊕0=1 · 1⊕1=0 · 0⊕0=0 · 0⊕1=1 · 1⊕1=0 ⟹ ได้ 110010 ตรงกับต้นฉบับ ✓

WE DO ทำด้วยกันครับ นี่คือตัวอย่างจากสไลด์เลย

> (a) Convert binary $1011\,0110$ to Gray. (b) Convert Gray $1011\,0110$ to binary.

บิตชุดเดียวกันเป๊ะ แต่โจทย์ถามคนละทิศ — ต้องได้คนละคำตอบครับ

1

`1 0 1 1 0 1 1 0`

`? ? ? ? ? ? ? ?`

(a) ขาไป — คู่จากแถวบนล้วน

(a) binary ⟹ Gray

MSB = ... · แล้ว XOR 7 คู่: .....

2

`1 0 1 1 0 1 1 0`

`? ? ? ? ? ? ? ?`

(b) ขากลับ — ทบต่อกัน

(b) Gray ⟹ binary

MSB = ... · แล้วทบทีละก้าว: .....

เฉลย WE DO — เติมทุกช่องพร้อมเหตุผล

(a) binary $1011\,0110$ **⟹ Gray**

ขั้น 1 — ลอก MSB เพราะกฎข้อ 1 ⟹ Gray บิตแรก = **1**

ขั้น 2 — XOR ทีละคู่จาก “แถว binary” ล้วน 8 บิตให้ 7 คู่ครับ ไล่จากซ้ายไปขวา:

$$1 \oplus 0 = 1 \;\cdot\; 0 \oplus 1 = 1 \;\cdot\; 1 \oplus 1 = 0 \;\cdot\; 1 \oplus 0 = 1 \;\cdot\; 0 \oplus 1 = 1 \;\cdot\; 1 \oplus 1 = 0 \;\cdot\; 1 \oplus 0 = 1$$

ขั้น 3 — ต่อกัน 1 กับ 1101101 ⟹ **1110 1101** (8 บิตเท่าเดิม ✓)

ขั้น 4 — ตรวจด้วยการแปลงกลับ เอา 11101101 มาทบต่อกัน: 1 · 1⊕1=0 · 0⊕1=1 · 1⊕0=1 · 1⊕1=0 · 0⊕1=1 · 1⊕0=1 · 1⊕1=0 ⟹ 10110110 ตรงกับโจทย์ ✓

(b) Gray $1011\,0110$ **⟹ binary**

ขั้น 1 — ลอก MSB เพราะกฎข้อ 1 ของขากลับก็บอกเหมือนกันว่าบิตซ้ายสุดเท่ากัน ⟹ binary บิตแรก = **1**

ขั้น 2 — ทบทีละก้าว: ผลลัพธ์ตัวก่อนหน้า XOR Gray ตัวถัดไป ทำไมต้องทีละก้าว เพราะแต่ละก้าวต้องรอผลของก้าวก่อนหน้า ข้ามไม่ได้ครับ:

$$b_2 = 1 \oplus 0 = 1 \;\cdot\; b_3 = 1 \oplus 1 = 0 \;\cdot\; b_4 = 0 \oplus 1 = 1 \;\cdot\; b_5 = 1 \oplus 0 = 1 \;\cdot\; b_6 = 1 \oplus 1 = 0 \;\cdot\; b_7 = 0 \oplus 1 = 1 \;\cdot\; b_8 = 1 \oplus 0 = 1$$

ขั้น 3 — ต่อกัน ⟹ 1101 1011

ขั้น 4 — ตรวจด้วยการแปลงไป เอา 11011011 ทำขาไป: 1 · 1⊕1=0 · 1⊕0=1 · 0⊕1=1 · 1⊕1=0 · 1⊕0=1 · 0⊕1=1 · 1⊕1=0 ⟹ 10110110 ตรงกับโจทย์ ✓

ตอบ: (a) $1011\,0110_2 = 1110\,1101_{\text{Gray}}$ **· (b)** $1011\,0110_{\text{Gray}} = 1101\,1011_2$ **— บิตชุดเดียวกันแต่ได้คนละคำตอบจริง ๆ ครับ**

“The two directions use different rules, so the same eight bits give two different results.”

YOU DO ตาน้องครับ

Y3.1

> Convert $1000\,0000_2$ to Gray code.

ข้อนี้ดูง่ายแต่มีกับดักครับ

เฉลย Y3.1

ขั้น 1 — ลอก MSB $\Longrightarrow$ Gray เริ่มด้วย **1** เพราะกฎข้อ 1

ขั้น 2 — XOR 7 คู่จากแถว binary ทำไมได้ 7 คู่ เพราะ 8 บิตวางคู่ติดกันได้ $8-1=7$ คู่ครับ:

$$1\oplus 0=1\ \cdot\ 0\oplus 0=0\ \cdot\ 0\oplus 0=0\ \cdot\ 0\oplus 0=0\ \cdot\ 0\oplus 0=0\ \cdot\ 0\oplus 0=0\ \cdot\ 0\oplus 0=0$$

ขั้น 3 — ต่อกัน 1 กับ 1000000 $\Longrightarrow$ **1100 0000** (ต้องได้ 8 บิต ห้ามตอบ 11 เฉย ๆ แล้วตัดศูนย์ท้ายทิ้งครับ)

ขั้น 4 — ตรวจด้วยการแปลงกลับ 11000000 ทบต่อกัน: $1\cdot 1\oplus 1{=}0\cdot 0\oplus 0{=}0\cdot 0\oplus 0{=}0\cdot 0\cdot 0\cdot 0\cdot 0$ $\Longrightarrow$ 10000000 ตรงกับโจทย์ ✓

ตอบ: $1000\,0000_2 = 1100\,0000_{\text{Gray}}$

“Copy the MSB, then XOR each adjacent pair: 1000 0000 becomes 1100 0000 in Gray code.”

Y3.2

> Convert 0101_{Gray} to base 2, and state its decimal value.

ขากลับครับ แล้วอย่าลืมว่า Gray เอาไปคูณน้ำหนักตรง ๆ ไม่ได้

เฉลย Y3.2

ขั้น 1 — ลอก MSB $\Longrightarrow$ binary บิตแรก = **0** เพราะบิตซ้ายสุดของสองรหัสเท่ากันเสมอ

ขั้น 2 — ทบทีละก้าว (ผลลัพธ์ตัวก่อน XOR Gray ตัวถัดไป)

$$b_2 = b_1 \oplus g_2 = 0 \oplus 1 = 1$$

$$b_3 = b_2 \oplus g_3 = 1 \oplus 0 = 1$$

$$b_4 = b_3 \oplus g_4 = 1 \oplus 1 = 0$$

ขั้น 3 — ได้เลขฐานสอง $\Longrightarrow$ 0110

ขั้น 4 — เพิ่งแปลงเสร็จค่อยคิดค่าฐานสิบได้ ตอนนี้มันเป็นเลขฐานสองจริง ๆ แล้ว น้ำหนักใช้ได้: $0110_2 = 4+2 = \mathbf{6}$ (ถ้าเผลอเอา Gray 0101 ไปคิดตรง ๆ จะได้ 5 ซึ่ง**ผิด**ครับ)

ขั้น 5 — ตรวจด้วยวิธีที่สอง เอา 0110 ทำขาไป: $0 \cdot 0 \oplus 1 = 1 \cdot 1 \oplus 1 = 0 \cdot 1 \oplus 0 = 1 \Longrightarrow 0101$ ตรงกับโจทย์ ✓ และเทียบกับตาราง 0–15 ข้างบน แถว decimal 6 มี Gray = 0101 พอดี ✓

ตอบ: $0101_{\text{Gray}} = 0110_2 = 6_{10}$

“Gray 0101 converts to binary 0110, which is decimal 6.”

Y3.3

> A 4-bit shaft encoder moves from position 7 to position 8. How many bits change if the disc is coded in ordinary binary? How many change if it is coded in Gray?

ข้อความรู้ครับ ตอบพร้อมเหตุผล

เฉลย Y3.3

ขั้น 1 — เขียนรหัสของทั้งสองตำแหน่งออกมาก่อน เพราะจะนับบิตที่เปลี่ยนได้ต้องเห็นทั้งคู่:

binary: $7 = 0111$ และ $8 = 1000$

Gray: $7 = 0100$ และ $8 = 1100$

ขั้น 2 — เทียบทีละตำแหน่ง (นี่คือ XOR นั่นเอง) เพราะบิตที่ต่างกันจะให้ผล XOR เป็น 1:

binary: $0111 \oplus 1000 = 1111 \Longrightarrow$ มี 1 อยู่ **4 ตัว** $\Longrightarrow$ เปลี่ยน **4 บิต**

Gray: $0100 \oplus 1100 = 1000 \Longrightarrow$ มี 1 อยู่ **1 ตัว** $\Longrightarrow$ เปลี่ยน **1 บิต**

ขั้น 3 — บอกว่าทำไมถึงสำคัญ ตอน binary เปลี่ยน 4 บิตพร้อมกัน ถ้าเซนเซอร์เหลื่อมกันนิดเดียว ค่าที่อ่านได้เป็นไปได้ตั้งหลายแบบ เช่น 1111 (=15) หรือ 0000 (=0) ซึ่ง**อยู่คนละฟากของจานเลย** ส่วน Gray แย่ที่สุดคืออ่านได้ 0100 (=7) หรือ 1100 (=8) คือคลาดไปแค่ช่องเดียวครับ

ตอบ: binary เปลี่ยน 4 บิต · Gray เปลี่ยน 1 บิต — และนี่คือเหตุผลทั้งหมดที่ shaft encoder ใช้ Gray

“In binary, 0111 to 1000 changes four bits at once; in Gray, 0100 to 1100 changes only one.”

4 ASCII — เขียนตัวอักษรด้วย 0 กับ 1

ตะขอเกี่ยว

สองหัวข้อที่แล้วเราเขียน**ตัวเลข**ให้เป็น 0 กับ 1 ได้แล้วครับ ทั้งแบบทีละหลัก (BCD) และแบบเปลี่ยนทีละบิต (Gray)

สมมติว่าถ้าน้องต้องส่งคำว่า “KEY” ไปตามสายละครับ — สายไฟรู้จักแค่แรงดันสูงกับต่ำ ไม่รู้จักตัวอักษรเลย น้องจะบอกมันยังไงว่านี่คือตัว K ครับ

ของใกล้ตัว: ตัวเลขบนปุ่มกุญแจล็อกรหัส

กุญแจล็อกรหัสหมุนตัวเลขบนแป้นครับ ตัวล็อกไม่ได้รู้จักคำว่า “เปิด” หรือ “ปิด” มันรู้จักแค่**ตัวเลขที่ตกลงกันไว้ล่วงหน้า** — ถ้าเราสองคนตกลงกันว่า “เลข 5 แปลว่าเปิดประตูหน้า” แค่นั้นก็สื่อสารกันได้แล้วครับ

ASCII ก็คือ**ตารางที่คนทั้งโลกตกลงกันว่าตัวอักษรไหนใช้เลขอะไร** เท่านั้นเองครับ ไม่มีสูตรคำนวณอะไรทั้งสิ้น เป็นแค่การตกลงกัน

DEFINITION · ASCII

ASCII (American Standard Code for Information Interchange) is a 7-bit code, so it has $2^7 = 128$ characters, numbered 00h to 7Fh. The most significant bit is therefore always 0 when it is stored in a byte.

พี่แปลว่า: **ASCII ใช้ 7 บิต จึงมีตัวอักษรได้ 128 ตัว เลขที่ 0 ถึง 127 (เขียนเป็นฐานสิบหกคือ 00h ถึง 7Fh)** เวลาเก็บลงไบต์ 8 บิต บิตบนสุดจึงเป็น 0 เสมอครับ

ตารางแบ่งเป็น 2 ครึ่ง

00h–1Fh · 32 ตัวแรก = control codes

พิมพ์ออกมาไม่ได้ครับ เป็นคำสั่งควบคุม เช่น NUL (ว่าง), ESC (หนี), CR (กลับต้นบรรทัด), LF (ขึ้นบรรทัดใหม่), BEL (ทำให้เครื่องส่งเสียง)

20h–7Fh · 96 ตัวหลัง = printable

พิมพ์ออกมาเห็นได้ครับ เริ่มที่ 20h คือช่องว่าง (space) แล้วต่อด้วยเครื่องหมายวรรคตอน ตัวเลข ตัวพิมพ์ใหญ่ ตัวพิมพ์เล็ก

หมุดหมายที่ควรจำ — จำแค่ 3 อันนี้พอ

ตัวเลข 0 ถึง 9	ตัวพิมพ์ใหญ่ A ถึง Z	ตัวพิมพ์เล็ก a ถึง z
30h – 39h	41h – 5Ah	61h – 7Ah

ความสัมพันธ์ที่ช่วยประหยัดการจำ

ตัวพิมพ์เล็ก = ตัวพิมพ์ใหญ่ + 20h · **รหัสของตัวเลข '7' = 30h + 7 = 37h**

จำแค่จุดเริ่มของ 3 กลุ่ม แล้วบวกระยะห่างเอาครับ ไม่ต้องท่องทั้งตาราง 128 ช่อง

กับดัก · ตัวอักษร '7' ไม่ใช่ตัวเลข 7

อันนี้คือกับดักคลาสสิกของ ASCII ครับ — **ตัวเลข 7 ที่เอาไปบวกลบ** เก็บเป็น 0000111_2 แต่ **ตัวอักษร '7' ที่พิมพ์บนหน้าจอ** เก็บเป็น $37h = 0110111_2$ ครับ

เพราะฉะนั้นถ้าเอา '7' กับ '2' มาบวกกันตรง ๆ จะไม่ได้ 9 นะครับ ต้อง**ลบ 30h ออกก่อน** เพื่อดึงค่าตัวเลขจริงออกมา — นี่คือเหตุผลที่โปรแกรมทุกภาษาต้องมีฟังก์ชันแปลงข้อความเป็นตัวเลข

เทียบของเก่า · EP01

EP01 น้องแปลงเลขฐานสิบหกเป็นฐานสองด้วยการ**จับกลุ่มทีละ 4 บิต**ใช่ไหมครับ — ตรงนี้เอามาใช้ต่อได้เลย: $44h \Rightarrow 4 = 0100$ และ $4 = 0100 \Longrightarrow 01000100$ แล้วถ้าอยากได้ 7 บิตก็ตัด 0 นำหน้าออกตัวเดียวเป็น 1000100 ครับ

I DO พี่ทำให้ดูครับ

ตัวอย่างที่พี่ทำ — โจทย์พี่แต่งเอง · ใช้ตาราง Reference สไลด์หน้า 37

> Give the ASCII code for the letter 'D' in hexadecimal, in decimal, and as 7 bits.

แปล: จงหารหัส ASCII ของตัวอักษร D ทั้งแบบฐานสิบหก ฐานสิบ และแบบ 7 บิตครับ

วิธีคิดของพี่

ขั้น 1 — หาจุดเริ่มของกลุ่มก่อน เพราะเราจำแค่จุดเริ่ม: ตัวพิมพ์ใหญ่เริ่มที่ **'A' = 41h**

ขั้น 2 — นับระยะห่าง เพราะตัวอักษรเรียงติดกันไม่มีเว้น: A, B, C, D $\Longrightarrow$ D ห่างจาก A อยู่ **3 ตัว**

ขั้น 3 — บวกระยะเข้าไป $41h + 3 = \mathbf{44h}$

ขั้น 4 — แปลงเป็นฐานสิบ $44h = 4 \times 16 + 4 = \mathbf{68}$

ขั้น 5 — แปลงเป็นบิต โดยจับกลุ่ม 4 บิตต่อหนึ่งหลักฐานสิบหก (วิธีจาก EP01) $4 \Rightarrow 0100$ ทั้งสองหลัก $\Longrightarrow$ $01000100_2 \Longrightarrow$ ตัดศูนย์นำหน้าเหลือ 7 บิตคือ $\mathbf{1000100}$

ตอบ: 'D' = 44h = 68 ฐานสิบ = 1000100_2 **(7 บิต)**

“The ASCII code for 'D' is 44h, that is 68 decimal, or 1000100 in seven bits.”

WE DO ทำด้วยกันครับ

ทำด้วยกัน โจทย์พี่แต่งเอง

> Give the ASCII code for the character '5' in hexadecimal and as 7 bits. Explain why it is not 0000101 .

ระวังครับ นี่คือ**ตัวอักษร** '5' ไม่ใช่**ตัวเลข** 5

เฉลย WE DO

ขั้น 1 — ดูให้ออกว่าโจทย์ถามตัวอักษรหรือตัวเลข โจทย์ใส่เครื่องหมายคำพูดครอบไว้ว่า '5' แปลว่าเป็น**ตัวอักษรที่พิมพ์บนหน้าจอ**ครับ ไม่ใช่ค่าตัวเลขที่เอาไปบวกลบ

ขั้น 2 — หาจุดเริ่มของกลุ่มตัวเลข เพราะกลุ่มตัวเลขเริ่มที่ **'0' = 30h**

ขั้น 3 — บวกระยะ '5' ห่างจาก '0' อยู่ 5 ตัว $\Longrightarrow 30h + 5 = \mathbf{35h}$

ขั้น 4 — แปลงเป็นบิตทีละหลักฐานสิบหก $3 \Rightarrow 0011$ และ $5 \Rightarrow 0101 \Longrightarrow 00110101_2 \Longrightarrow$ 7 บิตคือ $\mathbf{0110101}$

ขั้น 5 — ตอบคำถามว่าทำไมไม่ใช่ 0000101 เพราะ 0000101_2 คือ**ค่าตัวเลข 5** ซึ่งในตาราง ASCII ตรงกับ 05h ที่เป็น control code ชื่อ ENQ ไม่ใช่ตัวอักษรที่พิมพ์ออกมาได้ครับ

ตอบ: '5' = 35h = 0110101_2 **— ต่างจากค่าตัวเลข 5 (**0000101_2**) อยู่ 30h พอดี**

“The character '5' is 35h, not 05h; the digit characters start at 30h.”

YOU DO ตาน้องครับ

Y4.1

> Give the ASCII code for the lower-case letter 'k' in hexadecimal and as 7 bits.

ใช้ความสัมพันธ์ “ตัวเล็ก = ตัวใหญ่ + 20h” ให้เป็นประโยชน์ครับ

เฉลย Y4.1

ขั้น 1 — หาตัวพิมพ์ใหญ่ก่อน เพราะจำจุดเริ่มของมันได้ 'A' = 41h และ 'K' ห่างจาก 'A' อยู่ 10 ตัว (A B C D E F G H I J K) $\Longrightarrow 41h + 0Ah = \mathbf{4Bh}$

ขั้น 2 — บวก 20h เพื่อเปลี่ยนเป็นตัวพิมพ์เล็ก เพราะในตาราง ASCII ตัวพิมพ์เล็กอยู่ห่างจากตัวพิมพ์ใหญ่เท่ากันหมดคือ 20h $\Longrightarrow 4Bh + 20h = \mathbf{6Bh}$

ขั้น 3 — แปลงเป็นบิตทีละหลักฐานสิบหก $6 \Rightarrow 0110$ และ $B \Rightarrow 1011 \Longrightarrow 01101011_2 \Longrightarrow$ 7 บิตคือ $\mathbf{1101011}$

ขั้น 4 — ตรวจด้วยวิธีที่สอง ลองนับจากจุดเริ่มตัวพิมพ์เล็กตรง ๆ: 'a' = 61h และ 'k' ห่างไป 10 ตัว $\Longrightarrow 61h + 0Ah = 6Bh$ ตรงกัน ✓ และแปลงเป็นฐานสิบได้ $6 \times 16 + 11 = 107$ ซึ่งอยู่ในช่วง 97–122 ของตัวพิมพ์เล็กพอดี ✓

ตอบ: 'k' = 6Bh = 107 ฐานสิบ = 1101011_2

“The ASCII code for 'k' is 6Bh, which is 1101011 in seven bits.”

ศัพท์สอบของหัวข้อนี้

English	แปลว่า	ข้อสอบเขียนมาแบบนี้
character code	รหัสตัวอักษร	Character Codes — ASCII in one page
control code	รหัสควบคุม (พิมพ์ไม่ออก)	the first 32 are control codes, never printed
printable character	ตัวอักษรที่พิมพ์ออกมาได้	the last 96 are printable
extended ASCII	ASCII ขยาย (8 บิต 256 ตัว)	Extended ASCII adds 128 more using 8 bits

5 Parity — บิตเดียวที่บอกว่า “ข้อมูลเพี้ยนหรือเปล่า”

ตะขอเกี่ยว

สามหัวข้อที่แล้วเราเขียนของให้เป็น 0 กับ 1 ได้หมดแล้วครับ — เลขทีละหลัก (BCD) ลำดับมุม (Gray) ตัวอักษร (ASCII) ทั้งสามอันมีอย่างหนึ่งเหมือนกันคือ **ถ้าบิตพลิกไปหนึ่งตัว รหัสก็ยังหน้าตาปกติ**

สมมติว่าถ้าน้องส่งรหัส BCD ของเลข 5 คือ 0101 ไปตามสาย แล้วปลายทางได้ 0111 มาละครับ — ปลายทางจะอ่านว่า 7 ทันทีโดยไม่สงสัยอะไรเลย เพราะ 0111 ก็เป็นรหัส BCD ที่ถูกกฎทุกประการ น้องจะทำยังไงให้ปลายทาง “เอะใจ” ได้ครับ

ของใกล้ตัว: นับดินสอก่อนใส่กล่อง

น้องยืมดินสอเพื่อนไป 6 แท้ง ใส่กล่องส่งคืน ระหว่างทางหล่นหายไป 1 แท่ง — **ถ้าไม่ได้บอกจำนวนกันไว้ก่อน ปลายทางเปิดกล่องเห็น 5 แท่งก็คิดว่าครบครับ** แต่ถ้าน้องเขียนติดฝากล่องไว้ว่า “ข้างในเป็นจำนวนคู่” เพื่อนนับได้ 5 ซึ่งเป็นคี่ ก็รู้ทันทีว่าหายไปแล้ว

Parity bit ก็คือกระดาษแผ่นนั้นครับ — เป็นบิตส่วนเกิน 1 บิต ที่แนบไปเพื่อบังคับให้ **จำนวนเลข 1 ทั้งชุด** เป็นคู่ (หรือคี่) ตามที่ตกลงกันไว้

DEFINITION · Parity Bit

A parity bit is attached to a group of bits to make the total number of 1s in the group always even (even parity) or always odd (odd parity). Any given system operates with even or odd parity, but not both. The parity bit can be attached to either end of the code.

พี่แปลว่า: **parity bit คือบิตแถม 1 ตัว ที่เติมเข้าไปเพื่อบังคับให้จำนวนเลข 1 ทั้งชุด เป็นคู่เสมอ (even parity) หรือคี่เสมอ (odd parity)** ระบบหนึ่งเลือกได้แบบเดียว ห้ามใช้ปนกันครับ และจะแปะไว้หัวหรือท้ายก็ได้ ขอแค่ตกลงกันไว้

ที่มา: ทำไมแค่นับเลข 1 ถึงจับ error ได้

ตรงนี้ต้องเข้าใจก่อนถึงจะจำวิธีตรวจได้แม่นครับ — **บิตหนึ่งตัวพลิกได้แค่ 2 แบบ** คือ 0 กลายเป็น 1 (จำนวน 1 เพิ่มขึ้น 1) หรือ 1 กลายเป็น 0 (จำนวน 1 ลดลง 1) ไม่ว่าทางไหน **จำนวน 1 ก็ขยับไป 1 หน่วยเสมอ**

และการบวกหรือลบ 1 หน่วย **ทำให้ความเป็นคู่-คี่พลิกด้านทันที**ครับ — คู่กลายเป็นคี่ คี่กลายเป็นคู่ นี่แหละคือเหตุผลทั้งหมดว่าทำไม parity ถึงใช้ได้

พลิก 1 บิต — จำนวน 1 ขยับ 1 หน่วย ⟹ คู่-คี่พลิกด้าน ⟹ จับได้

`1 1 0 1 1` → `1 0 0 1 1` จำนวน 1 จาก 4 (คู่) เหลือ 3 (คี่) ⟹ ผิดกฎ even parity ⟹ ฝั่งรับจับ

พลิก 2 บิต — จำนวน 1 ขยับ 2 หน่วย ⟹ คู่-คี่กลับมาเหมือนเดิม ⟹ จับไม่ได้

`1 1 0 1 1` → `1 0 1 1 1` จำนวน 1 ยังเป็น 4 (คู่) เหมือนเดิม ⟹ ผ่านการตรวจทั้งที่ข้อมูลเสียไ
ข้อมูลจริงเปลี่ยนจาก 1011 เป็น 0111 แต่ฝั่งรับไม่รู้เลยครับ

สรุปเป็นกฎเดียว: **parity จับได้เฉพาะเมื่อบิตพลิกเป็น "จำนวนคี่" เท่านั้น** (1, 3, 5 บิต)

วิธีหา parity bit — 2 ขั้นจบ

1 `1 0 1 1 0 1` นับได้ 4 ตัว (คู่)

นับจำนวนเลข 1 ในข้อมูล

เพราะ parity สนใจแค่ "จำนวน 1" อย่างเดียวครับ ไม่ได้สนใจว่าบิตไหนเป็นอะไร หรือข้อมูลมีค่าเท่าไร — นับ 1 ให้ครบแล้วดูว่าได้คู่หรือคี่

2 ตรงกับที่ต้องการ **เติม 0** | ยังไม่ตรง **เติม 1**

เติมบิตที่ทำให้ได้คู่-คี่ตามที่โจทย์สั่ง

เพราะการเติม 1 คือการเพิ่มจำนวน 1 ไปอีกหนึ่งตัว ซึ่งพลิกคู่-คี่ ส่วนการเติม 0 ไม่เปลี่ยนอะไรครับ: ถ้าจำนวนที่นับได้**ตรง**กับที่ต้องการอยู่แล้ว ⟹ เติม **0** · ถ้า**ยังไม่ตรง** ⟹ เติม **1**

ตาราง parity ของรหัส BCD ทั้ง 10 ตัว (จากสไลด์หน้า 38)

ดูตารางนี้แล้วสังเกตครับ: **คอลัมน์ even กับ odd เป็นตรงข้ามกันทุกแถวเป๊ะ ๆ** เพราะทั้งสองแบบตอบคำถามเดียวกันแต่คนละด้าน

รหัส BCD	จำนวนเลข 1	Even parity bit	Odd parity bit	ตรวจสอบ (even ⟹ รวมต้องเป็นคู่)
0000	0 (คู่)	**0**	**1**	0+0 = 0 คู่ ✓
0001	1 (คี่)	**1**	**0**	1+1 = 2 คู่ ✓
0010	1 (คี่)	**1**	**0**	1+1 = 2 คู่ ✓
0011	2 (คู่)	**0**	**1**	2+0 = 2 คู่ ✓
0100	1 (คี่)	**1**	**0**	1+1 = 2 คู่ ✓
0101	2 (คู่)	**0**	**1**	2+0 = 2 คู่ ✓
0110	2 (คู่)	**0**	**1**	2+0 = 2 คู่ ✓
0111	3 (คี่)	**1**	**0**	3+1 = 4 คู่ ✓
1000	1 (คี่)	**1**	**0**	1+1 = 2 คู่ ✓

รหัส BCD	จำนวนเลข 1	Even parity bit	Odd parity bit	ตรวจสอบ (even ⟹ รวมต้องเป็นคู่)
1001	2 (คู่)	**0**	**1**	2+0 = 2 คู่ ✓

ทางลัด · ท่องประโยคเดียวจบ

“Even parity: นับ 1 ได้คี่ ⟹ เติม 1 · นับ 1 ได้คู่ ⟹ เติม 0” และ **odd parity ตอบตรงข้ามกับ even เสมอ** ครับ

เพราะฉะนั้นเวลาสอบเจอโจทย์ odd ให้**คิดแบบ even ก่อน แล้วพลิกคำตอบ** ก็ได้ครับ เร็วกว่าและพลาดยากกว่า

ฝั่งรับตรวจยังไง — และทำไมถึง “แก้ไม่ได้”

ฝั่งรับทำแค่อย่างเดียวครับ: **นับจำนวน 1 ของทั้งชุด (รวม parity bit ด้วย) แล้วดูว่าตรงกฎที่ตกลงกันไหม** ถ้าตรง ⟹ ผ่าน ถ้าไม่ตรง ⟹ มี error

แต่ตรงนี้คือข้อจำกัดใหญ่ครับ — **ฝั่งรับรู้แค่ว่า “มีอะไรผิด” แต่ไม่มีทางรู้ว่าผิดบิตไหน**

ฝั่งรับได้ 00001 มา (ระบบ even parity) — นับ 1 ได้ 1 ตัว เป็นคี่ ⟹ ผิดแน่นอน

แต่ “ผิดที่บิตไหน” มีความเป็นไปได้ 5 แบบพอ ๆ กันหมด

1 0 0 0 1	0 1 0 0 1	0 0 1 0 1	0 0 0 1 1	0 0 0 0 0
ถ้าผิดบิตที่ 1	ถ้าผิดบิตที่ 2	ถ้าผิดบิตที่ 3	ถ้าผิดบิตที่ 4	ถ้าผิดบิตที่ 5

ทั้ง 5 แบบผ่านกฎ even parity หมด — parity ไม่มีข้อมูลพอจะเลือกได้ว่าอันไหนถูก

ทางออกเดียวคือขอให้ต้นทางส่งใหม่ — หรือใช้ Hamming code ที่จะสอนในหัวข้อถัดไป

นี่คือเหตุผลที่ตำราเขียนว่า parity **detects** an error แต่ไม่ใช่ **corrects** ครับ

กับดัก · 3 อย่างที่พลาดบ่อยที่สุด

1. อ่านโจทย์ให้ออกว่า even หรือ odd — โจทย์ข้อ 7 ของ Assignment เขียนว่า even แต่ข้อ 8 เขียนว่า odd ครับ สลับกันคือผิดหมดทั้งข้อ

2. ตอนตรวจต้องนับ parity bit รวมเข้าไปด้วย ถ้านับเฉพาะข้อมูลจะได้คำตอบตรงข้ามเสมอ

3. อย่าตอบว่า “แก้ให้ถูกเป็น ...” — parity บอกได้แค่ผิด/ไม่ผิด ถ้าโจทย์ให้ระบุตำแหน่งด้วย แปลว่าโจทย์นั้นเป็น Hamming ไม่ใช่ parity ครับ

เทียบของเก่า · Gray code

สังเกตความต่างของสองอันนี้ให้ดีครับ — **Gray code พยายาม “ลดโอกาสเกิด” error** ด้วยการจัดรหัสให้เปลี่ยนทีละบิต ส่วน **parity ยอมให้ error เกิดได้ แล้วค่อย “จับ” ทีหลัง** คนละยุทธศาสตร์กันเลย และในระบบจริงใช้ทั้งสองอย่างพร้อมกันได้ครับ

ศัพท์สอบของหัวข้อนี้

English	แปลว่า	ข้อสอบเขียนมาแบบนี้
parity bit	บิตพาริตี	Assign the proper even parity bit to the following code groups
even / odd parity	พาริตีคู่ / พาริตีคี่	An odd parity system receives the following code groups
error detection	การตรวจจับข้อผิดพลาด	a parity bit as a means for bit error detection
single bit error	ข้อผิดพลาดบิตเดียว	Determine which groups (if any) contain a single bit error
cannot check for two errors	ตรวจสองบิตพร้อมกันไม่ได้	cannot check for two errors in one group

I DO พี่ทำให้ดูครับ

ตัวอย่างที่พี่ทำ โจทย์พี่แต่งเอง · ปูทางไป Assignment ข้อ 7

> Assign the proper even parity bit to 1010011_2.

แปล: จงเติม parity bit แบบคู่ให้กับ 1010011 ครับ

วิธีคิดของพี่

ขั้น 1 — อ่านโจทย์ว่าเป็น even หรือ odd โจทย์เขียน **even** ⟹ เป้าหมายคือทำให้จำนวน 1 ทั้งชุด**เป็นคู่**

ขั้น 2 — นับจำนวน 1 ในข้อมูล ไล่ทีละตัวเลขครับ 1 0 1 0 0 1 1 ⟹ มี 1 อยู่ที่ตำแหน่ง 1, 3, 6, 7 ⟹ **4 ตัว**

ขั้น 3 — เทียบกับเป้าหมาย 4 เป็นจำนวน**คู่**อยู่แล้ว ตรงกับ even ที่ต้องการ เพราะฉะนั้น**ไม่ต้องเพิ่มอะไรเข้าไป** ⟹ parity bit = **0**

ขั้น 4 — เขียนคำตอบเป็นชุดเต็ม วางไว้หน้าสุด (โจทย์ในสไลด์วางไว้หน้า) ⟹ **0 1010011** รวม 8 บิต

ขั้น 5 — ตรวจด้วยวิธีที่สอง นับ 1 ของชุดเต็ม 01010011 ⟹ 0+4 = 4 ตัว เป็นคู่ ✓ ผ่านกฎ even parity

ตอบ: even parity bit = 0 ⟹ ชุดเต็มคือ $0\,1010011$

“The group already contains four 1s, an even number, so the even parity bit is 0.”

WE DO ทำด้วยกันครับ

ทำด้วยกัน โจทย์พี่แต่งเอง · ปูทางไป Assignment ข้อ 8

> (a) Assign the proper odd parity bit to 0110010_2. (b) An odd parity system receives $1101\,0101$. Does it contain a single bit error?

(b) คือตัวอย่าง Q.2 จากสไลด์หน้า 40 ครับ ระวังว่าโจทย์ในสไลด์เป็น even แต่พี่เปลี่ยนเป็น odd

1

`0 1 1 0 0 1 0`
นับ 1 ได้ ? ตัว

(a) นับ 1 ก่อน

จำนวน 1 = ... ⟹ เป็น ..... ⟹ odd parity bit = ...

2

`1 1 0 1 0 1 0 1`
นับทั้งชุดที่รับมา

(b) ฝั่งรับนับทั้งชุด

จำนวน 1 = ... ⟹ ระบบ odd ต้องการ ..... ⟹ สรุปว่า .....

เฉลย WE DO — เติมทุกช่องพร้อมเหตุผล

(a) ขั้น 1 — นับ 1 ในข้อมูล 0110010 เพราะ parity สนใจแค่จำนวน 1: มี 1 ที่ตำแหน่ง 2, 3, 6 ⟹ **3 ตัว**

(a) ขั้น 2 — 3 เป็นจำนวนคี่ และระบบต้องการให้ผลรวมเป็น**คี่**อยู่แล้ว ⟹ ตรงเป้าหมายพอดี เพราะฉะนั้น**ไม่ต้องเติม 1 ⟹ odd parity bit = 0**

(a) ขั้น 3 — ชุดเต็ม ⟹ 0 0110010 · ตรวจซ้ำ: นับ 1 ได้ 3 ตัว เป็นคี่ ✓

(b) ขั้น 1 — นับ 1 ของทั้งชุดที่รับมา รวม parity bit ด้วย ทำไมต้องรวม เพราะกฎ odd parity พูดถึง “ทั้งชุด” ไม่ได้พูดถึงเฉพาะข้อมูลครับ: 1 1 0 1 0 1 0 1 ⟹ มี 1 ที่ตำแหน่ง 1, 2, 4, 6, 8 ⟹ **5 ตัว**

(b) ขั้น 2 — เทียบกับกฎ ระบบเป็น odd parity ⟹ ต้องได้จำนวน**คี่** และ 5 ก็เป็นคี่ ⟹ **ผ่าน**

(b) ขั้น 3 — สรุปให้ตรงคำถาม โจทย์ถามว่ามี single bit error ไหม ⟹ **ไม่มีการตรวจพบ error** แต่ต้องเขียนให้ระวังคำครับ: **“ไม่พบ” ไม่ได้แปลว่า “ไม่มี”** — ถ้าพลิกไป 2 บิต ชุดนี้ก็จะยังผ่านเหมือนกัน

ตอบ: (a) odd parity bit = 0 ⟹ $0\,0110010$ **· (b) ไม่พบ single bit error เพราะจำนวน 1 เท่ากับ 5 ซึ่งเป็นคี่ตามที่ระบบต้องการ**

“(b) The group has five 1s, an odd number, so no single-bit error is detected.”

YOU DO ตาน้องครับ

Y5.1

> In an even parity scheme, what is the parity bit for 1101? Write the full 5-bit group with the parity bit at the left.

เฉลย Y5.1

ขั้น 1 — อ่านว่าเป็น even ⟹ เป้าหมายคือทำให้จำนวน 1 ทั้งชุดเป็น**คู่**

ขั้น 2 — นับ 1 ในข้อมูล 1101 ⟹ มี 1 ที่ตำแหน่ง 1, 2, 4 ⟹ **3 ตัว เป็นคี่**

ขั้น 3 — ยังไม่ตรงเป้าหมาย จึงต้องเติม 1 เพราะการเติม 1 จะทำให้จำนวนขยับจาก 3 เป็น 4 ซึ่งเป็นคู่ ⟹ **parity bit = 1**

ขั้น 4 — เขียนชุดเต็มโดยวาง parity ไว้ซ้ายสุด ⟹ **1 1101**

ขั้น 5 — ตรวจด้วยวิธีที่สอง นับ 1 ของ 11101 ⟹ 4 ตัว เป็นคู่ ✓ ผ่านกฎ even parity

ตอบ: even parity bit = 1 ⟹ ชุดเต็มคือ 1 1101

“1101 contains three 1s, an odd number, so the even parity bit must be 1.”

Y5.2

> A system uses even parity. The code word 1 1011 is sent, but bits 2 and 3 are both flipped during transmission. Show what the receiver gets, and explain whether the error is detected.

ข้อนี้คือหัวใจของข้อจำกัดของ parity ครับ

เฉลย Y5.2

ขั้น 1 — เขียนชุดที่ส่งออกไปก่อน เพราะต้องมีตัวตั้งต้นถึงจะพลิกบิตได้: ชุดเต็มคือ 1 1 0 1 1 (parity bit = 1 อยู่ซ้ายสุด) ⟹ นับ 1 ได้ **4 ตัว เป็นคู่** ⟹ ถูกกฎ even parity ✓

ขั้น 2 — พลิกบิตที่ 2 และบิตที่ 3 เพราะโจทย์สั่งมา: บิตที่ 2 คือ 1 ⟹ กลายเป็น 0 · บิตที่ 3 คือ 0 ⟹ กลายเป็น 1 ⟹ ฝั่งรับได้ **1 0 1 1 1**

ขั้น 3 — ฝั่งรับนับ 1 ของชุดที่ได้มา 10111 ⟹ มี 1 ที่ตำแหน่ง 1, 3, 4, 5 ⟹ **4 ตัว เป็นคู่**

ขั้น 4 — เทียบกับกฎ 4 เป็นคู่ ตรงตามกฎ even parity ทุกประการ ⟹ **ฝั่งรับสรุปว่า “ไม่มี error”** ทั้งที่ข้อมูลจริงเปลี่ยนจาก 1011 เป็น 0111 ไปแล้ว

ขั้น 5 — อธิบายว่าทำไมถึงหลุด เพราะพลิกไป **2 บิต** จำนวน 1 ลดลง 1 แล้วเพิ่มขึ้น 1 **สุทธิแล้วไม่ขยับ** ⟹ ความเป็นคู่-คี่จึงไม่เปลี่ยน ⟹ parity มองไม่เห็น

ตอบ: ฝั่งรับได้ 10111 และตรวจไม่พบ error — เพราะ parity จับได้เฉพาะบิตพลิกจำนวนคี่เท่านั้น

“A parity bit provides for the detection of a single bit error but cannot check for two errors in one group.”

Y5.3

> An even parity system receives 0 1100 and 1 0110. Which group (if any) contains an error?

ข้อฝึกอ่านโจทย์ครับ ต้องนับทั้งชุด

เฉลย Y5.3

ขั้น 1 — ชุดแรก 01100 นับ 1 ทั้งชุด เพราะกฎพูดถึงทั้งชุดรวม parity bit: มี 1 ที่ตำแหน่ง 2, 3 ⟹ **2 ตัว เป็นคู่** ⟹ ตรงกฎ even parity ⟹ **ไม่พบ error**

ขั้น 2 — ชุดที่สอง 10110 นับ 1 ทั้งชุด มี 1 ที่ตำแหน่ง 1, 3, 4 ⟹ **3 ตัว เป็นคี่** ⟹ ขัดกับกฎ even parity ⟹ **พบ error**

ขั้น 3 — ตอบให้ตรงคำถาม และอย่าเผลอบอกตำแหน่ง โจทย์ถามว่า “ชุดไหนมี error” ตอบได้แค่ชื่อชุดครับ **ห้ามเดาว่าผิดบิตไหน** เพราะ parity ไม่มีข้อมูลตรงนั้น

ตอบ: ชุด 0 1100 **ผ่าน · ชุด** 1 0110 **มี error เพราะมีเลข 1 อยู่ 3 ตัวซึ่งเป็นคี่**

“The second group contains three 1s, an odd number, so it violates even parity and contains an error.”

6 CRC — เอาข้อมูลไปหาร แล้วส่งเศษไปด้วย

ตะขอเกี่ยว

หัวข้อที่แล้วน้องเห็นแล้วว่า parity มีรูโหว่ใหญ่ครับ — **พลิก 2 บิตเมื่อไหร่มันมองไม่เห็นเลย** และในสายจริง สัญญาณรบกวนมักมาเป็นชุด ไม่ได้มาทีละบิตสวย ๆ

สมมติว่าถ้าน้องต้องส่งข้อมูลยาว ๆ ทั้งก้อน แล้วอยากจับให้ได้แม้บิตพลิกติดกันหลายตัว ละครับ — บิตตรวจสอบแค่ตัวเดียวคงไม่พอแล้ว เราต้องการอะไรที่**ผูกกับข้อมูลทุกบิตพร้อมกัน** ครับ

ของใกล้ตัว: เศษที่เหลือจากการแบ่งของ

น้องมีดินสอ 17 แท่ง จะแบ่งใส่กล่องกล่องละ 5 แท่ง — ได้ 3 กล่องเต็ม **เหลือเศษ 2 แท่ง** ใช่ไหมครับ

ทีนี้ลองคิดกลับกัน: ถ้าน้อง**เอาเศษ 2 แท่งนั้นออกไปตั้งแต่แรก** เหลือ 15 แท่ง คราวนี้แบ่งกล่องละ 5 ได้**ลงตัวพอดี ไม่เหลือเศษ** — และถ้าปลายทางลองแบ่งดูแล้ว**ยังเหลือเศษ** แปลว่า**จำนวนดินสอเปลี่ยนไประหว่างทางแน่นอน** ครับ

CRC คือเรื่องนี้เป๊ะ ๆ ครับ — ฝั่งส่งจัดแจงให้ข้อมูล “หารลงตัว” ก่อนส่ง ฝั่งรับหารซ้ำ ถ้ายังลงตัวก็แปลว่ารอด ถ้าเหลือเศษก็แปลว่าเสีย

ฝั่งส่ง — จัดให้หารลงตัวก่อน

```
D = 110010
เติมศูนย์ 3 ตัว แล้วหารด้วย G
ส่งจริง = 110010 + เศษ
```

ฝั่งรับ — หารซ้ำด้วย G ตัวเดิม

```
เศษ = 0 ⟹ ข้อมูลรอด
เศษ ≠ 0 ⟹ ข้อมูลเสีย
ขอให้ส่งใหม่
```

ทั้งหมดของ CRC อยู่ในภาพนี้ครับ — “ทำให้ลงตัว” แล้ว “ตรวจว่ายังลงตัวไหม”

ก่อนอื่น: Modulo-2 คือ XOR ไม่ใช่การลบจริง

ตรงนี้ต้องย้ำหนัก ๆ ครับ เพราะเป็นจุดที่ทำให้คนหารผิดมากที่สุด — **การหารของ CRC ไม่มีการยืม ไม่มีตัวทด ไม่มีค่ามาก-น้อย** มันคือการ XOR ทีละหลักล้วน ๆ

Input	Bits	Output Bit	อ่านว่า
0	0	0	เหมือนกัน ได้ 0
0	1	1	ต่างกัน ได้ 1
1	0	1	ต่างกัน ได้ 1
1	1	0	เหมือนกัน ได้ 0

กับดัก · อันนี้พลาดกันเยอะที่สุด

แถวสุดท้าย **1 กับ 1 ได้ 0** ครับ — ไม่ใช่ 10 ไม่มีตัวทดขึ้นไปหลักหน้า

และแถวที่สอง **0 กับ 1 ได้ 1** — ไม่ต้องยืมจากหลักซ้าย ถ้าน้องเผลอทำเหมือนลบเลขฐานสองใน EP02 คำตอบจะเพี้ยนทั้งข้อครับ

สไลด์เรียกว่า **“binary addition with the carries discarded”** คือบวกแล้วทิ้งตัวทด ซึ่งให้ผลเหมือน XOR ทุกประการครับ

DEFINITION · Cyclic Redundancy Check

CRC is a widely used code for detecting one- and two-bit transmission errors. It can also detect multiple errors for a number of bits in sequence (burst errors). CRC is based on the division of two binary numbers using modulo-2 arithmetic.

พี่แปลว่า: **CRC คือรหัสตรวจข้อผิดพลาดที่ใช้การหารเลขฐานสองแบบ modulo-2 จับได้ทั้งบิตเดี่ยว บิตคู่ และบิตพลิกติดกันเป็นชุด (burst error)** ครับ แต่ยังไม่ได้จับได้ทุกกรณี — สไลด์เขียนไว้ว่า “not every possible error can be identified”

4 ขั้นฝั่งส่ง กับ 2 ขั้นฝั่งรับ (ตามสไลด์หน้า 43)

Sender — 4 ขั้น

1. ตกลง generator code **G** กันไว้ก่อน ทั้งสองฝั่งต้องรู้เหมือนกัน

2. เติมศูนย์ต่อท้ายข้อมูล r **ตัว** โดย r = ความยาวของ G ลบ 1

3. หารด้วย G แบบ modulo-2 (XOR ไม่มีการยืม)

4. เอาเศษไปแทนที่ศูนย์ที่เติมไว้ แล้วส่งออกไป

Receiver — 2 ขั้น

1. เอาชุดที่รับมา**ทั้งก้อน** หารด้วย G ตัวเดิม

2. เศษ = 0 $\Longrightarrow$ ไม่พบ error รับข้อมูลไว้ · เศษ ≠ 0 $\Longrightarrow$ มี error ขอให้ส่งใหม่

ฝั่งรับ**ไม่ต้องเติมศูนย์อีก**นะครับ เพราะช่องเศษถูกเติมไว้แล้วตั้งแต่ฝั่งส่ง

ที่มา · ทำไมวิธีนี้ถึงเวิร์ก

ขั้นที่ 2 กับ 4 ของฝั่งส่งทำงานร่วมกันแบบนี้ครับ: เติมศูนย์ r ตัวคือการ **“จองที่ว่าง”** ไว้ท้ายข้อมูล แล้วเอาเศษที่หารได้ไปวางในที่ว่างนั้น

ผลคือ**ชุดที่ส่งออกไปหารด้วย G ลงตัวพอดีเสมอ** — เพราะเรา XOR เศษกลับเข้าไป เศษก็หายไปพอดี (XOR ตัวเดียวกันสองครั้งได้ 0 ครับ)

เพราะฉะนั้นฝั่งรับแค่หารซ้ำ ถ้าได้เศษ 0 ก็แปลว่าบิตยังครบเหมือนตอนส่ง **CRC บอกได้แค่ว่า “มี error” — บอกไม่ได้ว่าอยู่บิตไหน และแก้ไม่ได้ครับ**

I DO พี่ทำเลขเล็ก ๆ ให้ดูก่อน ทุกรอบไม่ข้าม

ตัวอย่างที่พี่ทำ โจทย์พี่แต่งเอง · ปูทางไป Assignment ข้อ 9

> Determine the transmitted CRC for data (D) 1101 and generator code (G) 101.

แปล: หา CRC ที่ต้องส่งไป เมื่อข้อมูลคือ 1101 และตัวหารคือ 101 ครับ

ขั้นเตรียม — G = 101 ยาว 3 บิต $\Longrightarrow$ r = 3 − 1 = **2** $\Longrightarrow$ เติมศูนย์ต่อท้ายข้อมูล 2 ตัว ได้ตัวตั้งคือ **110100** และการหารจะมีทั้งหมด **4 รอบ** (เท่ากับจำนวนบิตของข้อมูล)

รอบ 1 · บิตนำ = 1 ⇒ XOR ด้วย G

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

รอบ 2 · บิตนำ = 1 ⇒ XOR ด้วย G

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

รอบ 3 · บิตนำ = 1 ⇒ XOR ด้วย G

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

รอบ 4 · บิตนำ = 0 ⇒ ข้าม

```
0 0 0 0 1 0
```

ไม่เปลี่ยน

สรุปฝั่งส่ง

ขั้น 1 — อ่านเศษจากผลลัพธ์รอบสุดท้าย ผลสุดท้ายคือ 000010 เศษคือ **2 บิตขวาสุด** $\Longrightarrow$ **CRC = 10**

ขั้น 2 — เอาเศษไปแทนที่ศูนย์ที่เติมไว้ เพราะขั้นที่ 4 ของฝั่งส่งสั่งไว้แบบนั้น: 1101 ต่อด้วย 10 $\Longrightarrow$ **ชุดที่ส่งจริง = 110110**

ตอบ: CRC = 10 · ชุดที่ส่งออกไปคือ 110110

WE DO ตัวอย่างจากสไลด์หน้า 48 และ 50 ครับ

ทำด้วยกัน สไลด์ Week 2 หน้า 48 และ 50

> Determine the transmitted CRC for the following byte of data (D) 11010011 and generator code (G) 1010. Verify that the remainder is 0. Then an error occurs in the second bit from the left; the received data is D' = 10010011010. Apply the CRC process to detect the error.

ข้อนี้ทำครบทั้งฝั่งส่งและฝั่งรับ 2 กรณีเลยครับ

ฝั่งส่ง — ตัวตั้งคือ 11010011000 (เติมศูนย์ 3 ตัว) มี 8 รอบ

รอบ 1 · บิตนำ = 1 ⇒ XOR ด้วย G

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

รอบ 2 · บิตนำ = 1 ⇒ XOR ด้วย G

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

รอบ 3 · บิตนำ = 1 ⇒ XOR ด้วย G

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

รอบ 4 · บิตนำ = 0 ⇒ ข้าม

```
0 0 0 0 1 0 1 1 0 0 0
```

ไม่เปลี่ยน

รอบ 5 · บิตนำ = 1 ⇒ XOR ด้วย G

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

รอบ 6 · บิตนำ = 0 ⇒ ข้าม

```
0 0 0 0 0 0 0 1 0 0 0
```

ไม่เปลี่ยน

รอบ 7 · บิตนำ = 0 ⇒ ข้าม

```
0 0 0 0 0 0 0 1 0 0 0
```

ไม่เปลี่ยน

รอบ 8 · บิตนำ = 1 ⇒ XOR ด้วย G

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

เฉลย WE DO ตอนที่ 1 — ฝั่งส่ง

ขั้น 1 — นับ r ก่อนลงมือ เพราะถ้าเติมศูนย์ผิดจำนวน คำตอบพังทั้งข้อครับ: G = 1010 ยาว 4 บิต ⟹ r = **3** ⟹ เติมศูนย์ 3 ตัว

ขั้น 2 — หารทีละรอบตามด้านบน ทุกรอบดูแค่ “บิตนำของรอบนั้นเป็น 1 หรือ 0” ถ้าเป็น 1 ก็ XOR ถ้าเป็น 0 ก็ข้าม — ไม่มีการเปรียบเทียบค่ามาก-น้อยเหมือนหารปกติครับ

ขั้น 3 — อ่านเศษ ผลสุดท้าย 00000000010 ⟹ เศษ 3 บิตขวาสุด = **010**

ตอบ: CRC = 010 · ชุดที่ส่งออกไปคือ 11010011010

ฝั่งรับ กรณีที่ 1 — ได้ 11010011010 มาครบถ้วน (8 รอบ)

รอบ 1 · บิตนำ = 1 ⇒ XOR ด้วย G

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

รอบ 2 · บิตนำ = 1 ⇒ XOR ด้วย G

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

รอบ 3 · บิตนำ = 1 ⇒ XOR ด้วย G

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

รอบ 4 · บิตนำ = 0 ⇒ ข้าม

```
0 0 0 0 1 0 1 1 0 1 0
```

ไม่เปลี่ยน

รอบ 5 · บิตนำ = 1 ⇒ XOR ด้วย G

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

รอบ 6 · บิตนำ = 0 ⇒ ข้าม

```
0 0 0 0 0 0 0 1 0 1 0
```

ไม่เปลี่ยน

รอบ 7 · บิตนำ = 0 ⇒ ข้าม

```
0 0 0 0 0 0 0 1 0 1 0
```

ไม่เปลี่ยน

รอบ 8 · บิตนำ = 1 ⇒ XOR ด้วย G

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

เศษ = 000 ⟹ เป็นศูนย์ทุกบิต ⟹ ไม่พบ error รับข้อมูลไว้ได้ครับ

ฝั่งรับ กรณีที่ 2 — บิตที่ 2 พลิก ได้ D' = 10010011010 มา (8 รอบ)

รอบ 1 · บิตนำ = 1 ⇒ XOR ด้วย G

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

รอบ 2 · บิตนำ = 0 ⇒ ข้าม

```
0 0 1 1 0 0 1 1 0 1 0
```

ไม่เปลี่ยน

รอบ 3 · บิตนำ = 1 ⇒ XOR ด้วย G

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

รอบ 4 · บิตนำ = 1 ⇒ XOR ด้วย G

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

รอบ 5 · บิตนำ = 1 ⇒ XOR ด้วย G

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

รอบ 6 · บิตนำ = 1 ⇒ XOR ด้วย G

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

รอบ 7 · บิตนำ = 0 ⇒ ข้าม

0 0 0 0 0 0 0 0 0 1 0

ไม่เปลี่ยน

รอบ 8 · บิตนำ = 0 ⇒ ข้าม

0 0 0 0 0 0 0 0 0 1 0

ไม่เปลี่ยน

เฉลย WE DO ตอนที่ 2 — ฝั่งรับ

ขั้น 1 — เทียบให้เห็นก่อนว่าอะไรเปลี่ยน ส่งไป 11010011010 แต่รับได้ 10010011010 ⟹ บิตที่ 2 จาก 1 กลายเป็น 0 ตามที่โจทย์บอกครับ

ขั้น 2 — หารด้วย G ตัวเดิม ไม่ต้องเติมศูนย์เพิ่ม เพราะช่องเศษถูกเติมมาแล้วจากฝั่งส่ง ถ้าเผลอเติมศูนย์อีกจะได้คำตอบผิดครับ

ขั้น 3 — อ่านเศษแล้วตัดสิน ผลสุดท้าย 00000000010 ⟹ เศษ = **010** ซึ่ง**ไม่ใช่ศูนย์**

ตอบ: กรณีไม่มี error เศษ = 000 ⟹ ผ่าน · กรณี D' = 10010011010 เศษ = 010 ≠ 0 ⟹ ตรวจพบ error

“The remainder is not zero, so an error is detected and a retransmission is requested.”

ทางลัด · เช็คงานตัวเองฟรี ๆ

หาเศษได้แล้ว**เอาชุดที่จะส่งไปหารซ้ำทันที**ครับ ต้องได้ศูนย์ทุกบิตเสมอ ถ้าไม่ได้ศูนย์แปลว่าหารผิดตั้งแต่ฝั่งส่ง — ตรวจแบบนี้ใช้เวลาไม่ถึงนาที แต่กันพลาดได้ทั้งข้อ

อีกอันหนึ่ง: **ความยาวของเศษต้องเท่ากับ r เสมอ** ถ้าได้สั้นกว่าให้เติม 0 นำหน้าจนครบ (เช่น $r = 3$ แล้วหารได้ 10 ต้องเขียนเป็น **010** ไม่ใช่ 10 ครับ)

ศัพท์สอบของหัวข้อนี้

English	แปลว่า	ข้อสอบเขียนมาแบบนี้
cyclic redundancy check (CRC)	การตรวจสอบด้วยส่วนซ้ำซ้อนเชิงวน	Determine the transmitted CRC for the following data
generator code (G)	รหัสตัวหาร	and generator code (G) 1101
modulo-2 addition	การบวกมอดุโล 2 (คือ XOR)	a method called modulo-2 addition can be used
discard carries	ทิ้งตัวทด	same as binary addition with the carries discarded
remainder	เศษ	Verify that the remainder is 0
burst error	ข้อผิดพลาดแบบต่อเนื่อง	multiple errors for a number of bits in sequence

English	แปลว่า	ข้อสอบเขียนมาแบบนี้
retransmission	การส่งซ้ำ	error, ask for a retransmission

YOU DO ตาน้องครับ

Y6.1

> Determine the transmitted CRC for data (D) 10110 and generator code (G) 1011, then verify at the receiver that the remainder is 0.

ข้อนี้มีเซอร์ไพรส์เล็ก ๆ ครับ ทำแล้วจะรู้

เฉลย Y6.1 — ฝั่งส่ง

ขั้น 1 — หา r G = 1011 ยาว 4 บิต $\Longrightarrow$ r = **3** $\Longrightarrow$ เติมศูนย์ 3 ตัว ได้ตัวตั้ง **10110000** และจะหาร **5 รอบ**

ขั้น 2 — หารทีละรอบ (ดูทุกรอบข้างล่างนี้) ทุกรอบตัดสินจากบิตนำอย่างเดียวครับ

รอบ 1 · บิตนำ = 1 ⇒ XOR ด้วย G

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

รอบ 2 · บิตนำ = 0 ⇒ ข้าม

```
0 0 0 0 0 0 0 0
```

ไม่เปลี่ยน

รอบ 3 · บิตนำ = 0 ⇒ ข้าม

```
0 0 0 0 0 0 0 0
```

ไม่เปลี่ยน

รอบ 4 · บิตนำ = 0 ⇒ ข้าม

```
0 0 0 0 0 0 0 0
```

ไม่เปลี่ยน

รอบ 5 · บิตนำ = 0 ⇒ ข้าม

```
0 0 0 0 0 0 0 0
```

ไม่เปลี่ยน

เฉลย Y6.1 — อ่านคำตอบ

ขั้น 3 — อ่านเศษ ผลสุดท้าย 00000000 $\Longrightarrow$ เศษ 3 บิตขวาสุด = **000**

ขั้น 4 — นี่คือเซอร์ไพรส์ของข้อนี้ เศษออกมาเป็น **ศูนย์ทั้งหมด** ครับ แปลว่าข้อมูลชุดนี้**หารด้วย** G **ลงตัวอยู่แล้วตั้งแต่แรก** — เกิดขึ้นได้จริงและไม่ใช่ความผิดพลาด เวลาเจอแบบนี้**ห้ามตกใจแล้วกลับไปหารใหม่**ครับ

ขั้น 5 — ชุดที่ส่งจริง ก็คือข้อมูลต่อด้วยเศษ $\Longrightarrow$ **10110000** (ยังต้องเขียนเศษ 3 บิตติดไปด้วยนะครับ ห้ามส่งแค่ 10110 เฉย ๆ)

ฝั่งรับ — หาร 10110000 ด้วย 1011 อีกครั้ง (5 รอบ)

รอบ 1 · บิตนำ = 1 ⇒ XOR ด้วย G

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

รอบ 2 · บิตนำ = 0 ⇒ ข้าม

0 0 0 0 0 0 0 0

ไม่เปลี่ยน

รอบ 3 · บิตนำ = 0 ⇒ ข้าม

0 0 0 0 0 0 0 0

ไม่เปลี่ยน

รอบ 4 · บิตนำ = 0 ⇒ ข้าม

0 0 0 0 0 0 0 0

ไม่เปลี่ยน

รอบ 5 · บิตนำ = 0 ⇒ ข้าม

0 0 0 0 0 0 0 0

ไม่เปลี่ยน

เฉลย Y6.1 — ฝั่งรับ

ขั้น 6 — อ่านเศษฝั่งรับ ผลสุดท้าย 00000000 ⟹ เศษ = **000** เป็นศูนย์ทุกบิต ✓

ตอบ: CRC = 000 · ชุดที่ส่งคือ 10110000 · ฝั่งรับหารแล้วเศษ = 000 ⟹ ไม่พบ error

“The remainder is zero at the receiver, so no error is detected.”

กับดักรวมของ CRC

1. เติมศูนย์ผิดจำนวน — r คือความยาว G **ลบ 1** ไม่ใช่ความยาว G ครับ G 4 บิตเติมศูนย์ 3 ตัว

2. ฝั่งรับเผลอเติมศูนย์อีกรอบ — ห้ามเด็ดขาด ชุดที่รับมาพร้อมหารได้เลย

3. ทำ XOR เป็นการลบ — เห็น 0 ลบ 1 แล้วยืมจากหลักซ้าย นั่นคือวิธีของ EP02 ไม่ใช่ของ CRC ครับ

4. หยุดหารเร็วเกินไป — ต้องหารให้ครบทุกรอบตามจำนวนบิตข้อมูล แม้บิตนำจะเป็น 0 ติดกันหลายรอบก็ต้องนับรอบไปเรื่อย ๆ ครับ

7 Hamming code — บอกได้ด้วยว่าผิดตรงไหน

ตะขอเกี่ยว

สองหัวข้อที่แล้ว parity กับ CRC ทำได้อย่างเดียวกันครับ คือบอกว่า **“มีอะไรผิด”** แล้วจบแค่นั้น ต้องขอให้ต้นทางส่งใหม่ทุกครั้ง

สมมติว่าถ้าการส่งใหม่ทำไม่ได้ล่ะครับ — เช่นข้อมูลจากดาวเทียมที่กว่าจะส่งกลับมาได้ใช้เวลานาน หรือหน่วยความจำในเครื่องที่อ่านผิดแล้วไม่มีต้นฉบับให้ขอ เราต้อง**ซ่อมเองที่ปลายทาง**ให้ได้ครับ และการจะซ่อมได้ ต้องรู้ก่อนว่า**บิตไหน**ที่พัง

ของใกล้ตัว: ทายว่ากุญแจอยู่ลิ้นชักใบไหน

สมมติมีลิ้นชัก 8 ใบ เรียงเบอร์ 0 ถึง 7 มีกุญแจซ่อนอยู่ใบเดียว น้องถามได้แต่คำถามที่ตอบว่า “ใช่/ไม่ใช่” เท่านั้น — **น้องต้องถามกี่คำถามถึงจะรู้แน่นอนว่าใบไหนครับ**

คำตอบคือ **3 คำถาม** ครับ และคำถามที่ฉลาดที่สุดคือถามแบบนี้:

คำถามที่ 1 — “เบอร์ลิ้นชักเป็นเลขคี่ไหม”
ตอบใช่ = บิตค่า 1 ของเบอร์เป็น 1 · ตอบไม่ใช่ = เป็น 0

คำถามที่ 2 — “เบอร์อยู่ในกลุ่ม 2-3 หรือ 6-7 ไหม”
ตอบใช่ = บิตค่า 2 ของเบอร์เป็น 1

คำถามที่ 3 — “เบอร์มากกว่าหรือเท่ากับ 4 ไหม”
ตอบใช่ = บิตค่า 4 ของเบอร์เป็น 1

คำตอบ 3 ข้อต่อกันคือเลขฐานสองของเบอร์ลิ้นชักพอดี

แต่ละคำถามถามถึง **บิตหนึ่งบิตของ “เลขที่อยู่”** — นี่คือหัวใจของ Hamming code ทั้งดวงครับ

Hamming code ทำแบบนี้เป๊ะครับ — มันแอบใส่ “คำถาม” ไว้ในข้อมูลหลายข้อ แต่ละข้อคือ parity bit หนึ่งตัวที่คุมตำแหน่งคนละกลุ่ม พอปลายทางตรวจแล้วพบว่าข้อไหนไม่ผ่านบ้าง **คำตอบทั้งชุดต่อกันก็กลายเป็นเลขที่อยู่ของบิตที่พังทันที**

DEFINITION · Hamming Code

Hamming adds several parity bits, each covering a different overlapping group of positions. The pattern of which checks fail points straight at the faulty bit, so it can be corrected.

พี่แปลว่า: **Hamming เติม parity หลายตัว แต่ละตัวคุมกลุ่มตำแหน่งที่ซ้อนทับกัน รูปแบบของ parity ที่ตรวจไม่ผ่านจะชี้ตรงไปที่บิตที่เสีย ทำให้แก้ได้**ครับ คำสำคัญคือ **overlapping** — กลุ่มต้องซ้อนกัน ถ้าแยกกันสนิทจะบอกตำแหน่งไม่ได้

ขั้นที่ 1 ของทุกข้อ: ต้องใช้ parity กี่บิต

$$2^p \geq d + p + 1$$

d = จำนวนบิตข้อมูล · p = จำนวนบิตพาริตี

ที่มาของสูตรนี้ไม่ยากเลยครับ — parity p ตัวให้คำตอบได้ 2^p แบบ และเราต้องการให้มันบอกได้ครบทุกกรณีคือ **“ไม่มีบิตไหนผิด”** 1 กรณี บวกกับ **“ผิดที่ตำแหน่งที่ 1 ถึงตำแหน่งที่** $d+p$**”** อีก $d+p$ กรณี รวมเป็น $d+p+1$ กรณี $\Longrightarrow$ ต้องมี $2^p \geq d+p+1$ ครับ

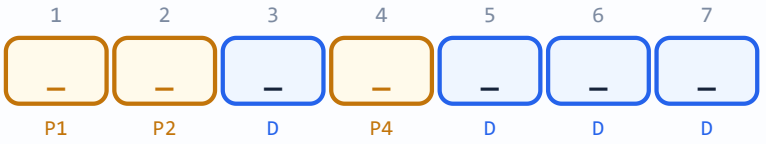
d (บิตข้อมูล)	ลอง p	2^p	$d+p+1$	ผ่านไหม	ความยาวรหัสรวม $n=d+p$
4	2	4	4+2+1 = 7	**ไม่ผ่าน**	—
	3	8	4+3+1 = 8	**ผ่านพอดี**	**7 บิต**
7	3	8	7+3+1 = 11	**ไม่ผ่าน**	—
	4	16	7+4+1 = 12	**ผ่าน**	**11 บิต**

จำ 2 บรรทัดนี้ไปสอบได้เลยครับ: $d=4 \Rightarrow p=3$ **(รหัส 7 บิต) และ** $d=7 \Rightarrow p=4$ **(รหัส 11 บิต)**

ขั้นที่ 2: บิตพาริตีอยู่ตำแหน่งกำลังของ 2 เสมอ

นับตำแหน่ง**เริ่มจาก 1** (ไม่ใช่ 0 นะครับ ตรงนี้ต่างจากที่เคยชิน) แล้ววางแบบนี้:

- **ตำแหน่ง 1, 2, 4, 8, 16 ...** (กำลังของ 2) $\Longrightarrow$ เก็บ **parity bit** เรียกว่า P1, P2, P4, P8
- **ตำแหน่งที่เหลือ 3, 5, 6, 7, 9, 10, 11 ...** $\Longrightarrow$ เก็บ **data bit** เรียงตามลำดับเดิมของข้อมูล



โครงรหัส 7 บิต (สำหรับข้อมูล 4 บิต) — ช่องส้มคือ parity ช่องฟ้าคือ data

ขั้นที่ 3: ตารางว่า “ใครคุมใคร” — และทำไมถึงคุมตามนั้น

RULE · Who checks whom

Parity bit at position k checks every position whose number contains k.

พี่แปลว่า: **Pk คุมทุกตำแหน่งที่เลขตำแหน่งนั้น “มีบิตค่า k เป็น 1”** ครับ

เพราะฉะนั้นวิธีดูคือ **เขียนเลขตำแหน่งเป็นเลขฐานสองก่อน** แล้วดูว่ามีบิตไหนติด 1 บ้าง เช่นตำแหน่งที่ 6 เขียนเป็น 0110 มีบิตค่า 4 กับบิตค่า 2 ติด ⟹ **ตำแหน่งที่ 6 ถูกคุมโดย P4 และ P2** ครับ

ตารางข้างล่างนี้คือตารางเดียวกับที่สไลด์หน้า 57 วาดไว้ แต่พี่เขียนใหม่ให้อ่านง่ายขึ้น — **แถว “เลขตำแหน่งฐานสอง” คือกุญแจของทั้งเรื่อง** ครับ:

รหัส 7 บิต (ข้อมูล 4 บิต)

ตำแหน่ง	1	2	3	4	5	6	7
เก็บอะไร	P	P	D	P	D	D	D
เลขตำแหน่งฐานสอง	001	010	011	100	101	110	111
P1	✓	·	✓	·	✓	·	✓
P2	·	✓	✓	·	·	✓	✓
P4	·	·	·	✓	✓	✓	✓

อ่านตารางนี้แนวตั้งก็ได้ครับ: คอลัมน์ตำแหน่งที่ 7 มีเครื่องหมายถูกทั้ง P1 P2 P4 เพราะ 7 เขียนเป็น 111 มีบิตติดครบทั้งสามตัว

รหัส 11 บิต (ข้อมูล 7 บิต) — อันนี้ใช้ตอนทำ Assignment ข้อ 10

ตำแหน่ง	1	2	3	4	5	6	7	8	9	10	11
เก็บอะไร	P	P	D	P	D	D	D	P	D	D	D
เลขตำแหน่งฐานสอง	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011
P1	✓	·	✓	·	✓	·	✓	·	✓	·	✓
P2	·	✓	✓	·	·	✓	✓	·	·	✓	✓

ตำแหน่ง	1	2	3	4	5	6	7	8	9	10	11
P4	·	·	·	✓	✓	✓	✓	·	·	·	·
P8	·	·	·	·	·	·	·	✓	✓	✓	✓

ความสวยงามของ Hamming: ทำไมผลบวกถึงชี้ตำแหน่งได้พอดี

ตรงนี้คือส่วนที่พี่อยากให้น้องเข้าใจมากที่สุดของทั้ง EP ครับ ไม่ใช่แค่ท่องวิธี

สมมติบิตที่ตำแหน่งที่ **6** พลิก ลองคิดตามทีละขั้นครับ:

1 ตำแหน่ง 6

0 1 1 0

8 4 2 1

เขียนเลขตำแหน่งเป็นฐานสองก่อน

เพราะกฎ “ใครคุมใคร” ผูกกับบิตของเลขตำแหน่งโดยตรงครับ: $6 = 0110_2 \Rightarrow$ มีบิตค่า 4 และบิตค่า 2 ติด 1 ส่วนบิตค่า 8 กับ 1 เป็น 0

2 P1 ผ่าน | P2 ไม่ผ่าน | P4 ไม่ผ่าน | P8 ผ่าน

ดูว่า parity ตัวไหนจะพัง

เพราะ parity ตัวหนึ่งจะพังก็ต่อเมื่อบิตที่พลิกอยู่ในกลุ่มที่มันคุมครับ: ตำแหน่ง 6 อยู่ในกลุ่มของ P2 และ P4 (ตามบิตที่ติด 1) $\Rightarrow$ **P2 กับ P4 พัง ส่วน P1 กับ P8 ยังผ่านปกติ**

3 2 + 4 = 6

ตรงกับตำแหน่งที่พลิกพอดี

บวกเลขตำแหน่งของ parity ที่พัง

เพราะเซตของ parity ที่คุมตำแหน่ง 6 คือชุดบิตที่ติด 1 ในเลข 6 พอดี การบวกกลับก็คือการประกอบเลขฐานสองกลับคืนมาครับ: $2 + 4 = 6$ $\Rightarrow$ **ชี้ตรงไปที่ตำแหน่งที่ 6**

4 ไม่มี parity ตัวไหนพังเลย ผลบวก = 0 = ไม่มี error

กรณีไม่มีบิตไหนผิด

เพราะ 0 ไม่ใช่เลขตำแหน่งของใครเลย (ตำแหน่งเริ่มนับที่ 1) ผลบวกเท่ากับ 0 จึงแปลว่า **ไม่พบข้อผิดพลาด** ได้อย่างไม่กำกวมครับ — นี่คือเหตุผลที่สูตรต้องบวก 1 เผื่อกรณีนี้ไว้

สรุปความสวยงามเป็นประโยคเดียว

“เซตของ parity ที่คุมตำแหน่ง x = เซตของบิตที่ติด 1 ในเลข x” — เพราะฉะนั้นเวลาเอาเลขตำแหน่งของ parity ที่พังมาบวกกัน มันคือการ**ประกอบเลข x กลับคืนมา** ครับ ผลบวกที่ได้เรียกว่า **syndrome** และมันคือเลขที่อยู่ของบิตที่พังพอดี ไม่ต้องเดา ไม่ต้องลองผิดลองถูก

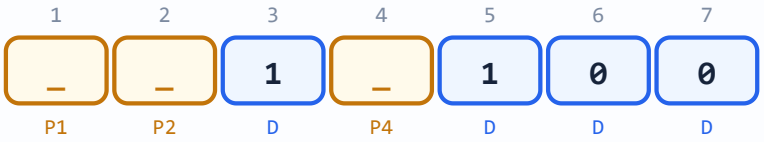
I DO พี่เข้ารหัสให้ดูครับ — ตัวอย่างจากสไลด์หน้า 59

ตัวอย่างที่พี่ทำ สไลด์ Week 2 หน้า 59

> Encode 4-bit data 1100 in Hamming code using even parity.

แปล: จงเข้ารหัส Hamming ให้ข้อมูล 4 บิต 1100 โดยใช้ even parity ครับ

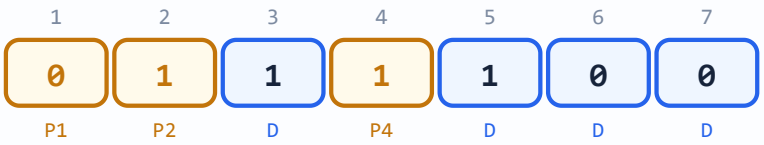
ขั้นเตรียม — $d = 4 \Longrightarrow$ จากตารางข้างบน $p = 3 \Longrightarrow$ รหัสยาว $n = 7$ บิต $\Longrightarrow$ parity อยู่ตำแหน่ง 1, 2, 4 · data อยู่ตำแหน่ง 3, 5, 6, 7



วางข้อมูล 1100 ลงในช่องข้อมูลตามลำดับ แล้วเหลือช่อง P ว่างไว้ก่อน

ขั้นคำนวณ parity ทีละตัว — แต่ละตัวนับเฉพาะ**บิตข้อมูล**ที่มันคุม (ยังไม่ต้องนับ parity ตัวอื่น เพราะยังไม่รู้ค่า) แล้วเติมให้จำนวน 1 เป็นคู่ตามกฎ even parity ครับ:

บิตพาริตี	คุมตำแหน่งที่เป็นข้อมูล	บิตในตำแหน่งนั้น	จำนวน 1	ต้องเติมอะไร
P1	3, 5, 7	1 1 0	2	**0**
P2	3, 6, 7	1 0 0	1	**1**
P4	5, 6, 7	1 0 0	1	**1**



เติม parity ครบแล้ว ได้รหัส Hamming สมบูรณ์

คำตอบและการตรวจ

ตอบ: 1100 เข้ารหัส Hamming (even parity) ได้ 0111100

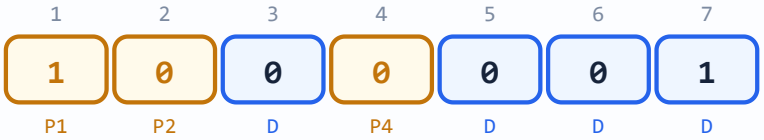
ตรวจด้วยวิธีที่สอง — เอารหัสที่ได้มาตรวจแบบฝั่งรับเลยครับ ต้องได้ผลบวกเท่ากับ 0: C1 นับตำแหน่ง 1,3,5,7 · C2 นับตำแหน่ง 2,3,6,7 · C4 นับตำแหน่ง 4,5,6,7 — ทั้งสามชุดต้องได้จำนวน 1 เป็นคู่หมด ถ้ามีชุดไหนเป็นคี่แปลว่าคำนวณ parity ผิดครับ

WE DO ถอดรหัสด้วยกันครับ — ตัวอย่างจากสไลด์หน้า 61

ทำด้วยกัน สไลด์ Week 2 หน้า 61

> Detect and correct the error from received Hamming code data 1000001 (even parity).

ฝั่งรับครับ ต้องหาว่าผิดตรงไหนแล้วแก้ให้ด้วย



รหัสที่รับมา 1000001 วางลงในตำแหน่ง 1 ถึง 7

ขั้นตรวจ — คราวนี้ต่างจากตอนเข้ารหัสตรงที่ **ต้องนับ parity bit รวมเข้าไปด้วย** เพราะฝั่งรับไม่รู้ว่าบิตไหนคือของจริงของปลอม มันแค่ตรวจว่ากฎยังเป็นจริงไหมครับ:

การตรวจ	ตำแหน่งที่ต้องนับ	บิตในตำแหน่งนั้น	จำนวน 1	ผลการตรวจ (even parity)
C1	1, 3, 5, 7	1 0 0 1	2	**คู่ ⟹ ผ่าน (C1 = 0)**
C2	2, 3, 6, 7	0 0 0 1	1	**คี่ ⟹ ผิด (C2 = 1)**
C4	4, 5, 6, 7	0 0 0 1	1	**คี่ ⟹ ผิด (C4 = 1)**

เฉลย WE DO

ขั้น 1 — ตรวจทีละกลุ่มตามตารางข้างบน ได้ผลว่า กลุ่มที่ **ไม่ผ่าน** คือ C2, C4 ส่วนกลุ่มที่ผ่านคือ C1 ครับ

ขั้น 2 — บวกเลขตำแหน่งของกลุ่มที่ไม่ผ่าน เพราะเลขตำแหน่งของ parity ที่พัง ประกอบกันเป็นเลขฐานสองของตำแหน่งที่เสียพอดี: 2 + 4 = 6 ⟹ **syndrome = 6**

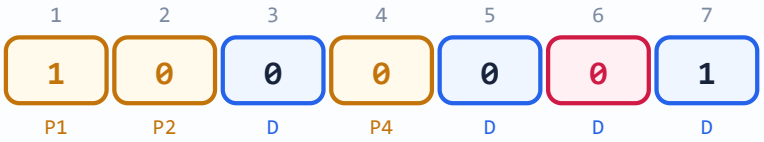
ขั้น 3 — แปลผล ผลบวกไม่เท่ากับ 0 ⟹ มี error และอยู่ที่ **ตำแหน่งที่ 6** (นับจากซ้าย เริ่มที่ 1)

ขั้น 4 — แก้โดยพลิกบิตนั้น เพราะบิตมีได้แค่ 0 กับ 1 การ “แก้” จึงคือการพลิกกลับ: ตำแหน่งที่ 6 เดิมเป็น 0 ⟹ เปลี่ยนเป็น 1 ⟹ **รหัสที่ถูกต้องคือ 1000011**

ขั้น 5 — ตรวจซ้ำหลังแก้ เอา 1000011 มาตรวจใหม่ทั้ง 3 กลุ่ม ต้องได้จำนวน 1 เป็นคู่หมด และ syndrome = 0 ✓ ถ้ายังไม่เป็น 0 แปลว่าหาตำแหน่งผิดครับ

ตอบ: มี error ที่ตำแหน่งที่ 6 ⟹ แก้แล้วได้ 1000011

“Checks C2,C4 fail, so the syndrome is 6 and the error is in bit position 6.”



ตำแหน่งที่ 6 (กรอบแดง) คือบิตที่ syndrome ชี้มา

กับดักรวมของ Hamming

1. นับตำแหน่งเริ่มที่ 1 ไม่ใช่ 0 — ถ้าเริ่มนับที่ 0 ตำแหน่งจะเลื่อนหมดทุกตัวครับ

2. ตอนเข้ารหัสไม่ต้องนับ parity ตัวอื่น แต่ตอนตรวจต้องนับทั้งหมด — สองขั้นนี้ใช้กติกาต่างกัน ตรงนี้พลาดกันเยอะที่สุด

3. บิตข้อมูลต้องเรียงตามลำดับเดิม — ข้อมูลตัวแรกไปตำแหน่ง 3 ตัวที่สองไปตำแหน่ง 5 ตัวที่สามไปตำแหน่ง 6 ห้ามสลับ

4. Hamming แก้ได้จริงแค่ “บิตเดียว” — ถ้าพลิก 2 บิต syndrome จะชี้ไปที่ตำแหน่งอื่น ที่ไม่ได้ผิดเลย แล้วจะยิ่งทำให้ข้อมูลพังหนักกว่าเดิม สไลด์เขียนไว้ว่า “Two errors in the same word will still fool it.” ครับ

ศัพท์สอบของหัวข้อนี้

English	แปลว่า	ข้อสอบเขียนมาแบบนี้
Hamming code	รหัสแฮมมิง	Encode 7-bit data 1011001 in Hamming code using even parity
error correction	การแก้ไขข้อผิดพลาด	the faulty bit can be corrected
parity bit position	ตำแหน่งของบิตพาริตี	Powers of two (1, 2, 4, 8) hold the parity bits
overlapping group	กลุ่มที่ซ้อนทับกัน	each covering a different overlapping group of positions
detect and correct	ตรวจจับและแก้ไข	Detect and correct the error from received Hamming code data

YOU DO ตาน้องครับ

Y7.1

> Encode 4-bit data 1011 in Hamming code using even parity.

เฉลย Y7.1

ขั้น 1 — หาจำนวน parity ก่อน $d = 4 \Longrightarrow$ ลอง $p = 3$: $2^3 = 8 \geq 4 + 3 + 1 = 8$ ผ่านพอดี $\Longrightarrow$ รหัสยาว **7 บิต**

ขั้น 2 — วางบิตลงตำแหน่ง เพราะ parity ต้องอยู่ตำแหน่งกำลังของ 2: ตำแหน่ง 1, 2, 4 = P · ตำแหน่ง 3, 5, 6, 7 = data เรียงตามลำดับ 1011 $\Longrightarrow$ ตำแหน่ง 3 = 1 · ตำแหน่ง 5 = 0 · ตำแหน่ง 6 = 1 · ตำแหน่ง 7 = 1

ขั้น 3 — คำนวณ parity ทีละตัว (นับเฉพาะบิตข้อมูลที่มันคุม)

บิตพาริตี	คุมตำแหน่งที่เป็นข้อมูล	บิตในตำแหน่งนั้น	จำนวน 1	ต้องเติมอะไร
P1	3, 5, 7	`1 0 1`	2	**0**
P2	3, 6, 7	`1 1 1`	3	**1**
P4	5, 6, 7	`0 1 1`	2	**0**

เฉลย Y7.1 (ต่อ)

ขั้น 4 — ประกอบรหัส เรียงตำแหน่ง 1 ถึง 7 $\Longrightarrow$ **0110011**

ขั้น 5 — ตรวจด้วยวิธีที่สอง ลองตรวจแบบฝั่งรับ: C1 (ตำแหน่ง 1,3,5,7), C2 (2,3,6,7), C4 (4,5,6,7) ต้องได้จำนวน 1 เป็นคู่ทั้งสามชุด $\Longrightarrow$ syndrome = 0 ✓ แปลว่าเข้ารหัสถูกต้องครับ

ตอบ: 1011 $\Longrightarrow$ 0110011

“The Hamming code for 1011 with even parity is 0110011.”

Y7.2

> The word 0110111 is received (even parity Hamming). Find the error position and correct it.

นี่คือรหัสจากข้อ Y7.1 ที่ถูกพลิกไปหนึ่งบิตครับ

เฉลย Y7.2

ขั้น 1 — ตรวจทีละกลุ่ม โดยนับ parity bit รวมเข้าไปด้วย (ต่างจากตอนเข้ารหัสนะครับ)

การตรวจ	ตำแหน่งที่ต้องนับ	บิตในตำแหน่งนั้น	จำนวน 1	ผลการตรวจ (even parity)
C1	1, 3, 5, 7	`0 1 1 1`	3	**คี่ ⟹ ผิด (C1 = 1)**
C2	2, 3, 6, 7	`1 1 1 1`	4	**คู่ ⟹ ผ่าน (C2 = 0)**
C4	4, 5, 6, 7	`0 1 1 1`	3	**คี่ ⟹ ผิด (C4 = 1)**

เฉลย Y7.2 (ต่อ)

ขั้น 2 — สรุปผลการตรวจ กลุ่มที่ **ไม่ผ่าน** คือ C1, C4 ส่วนกลุ่มที่ผ่านคือ C2 ครับ

ขั้น 3 — บวกเลขตำแหน่งของกลุ่มที่ไม่ผ่าน 1 + 4 = 5 ⟹ **syndrome = 5**

ขั้น 4 — แก้บิตที่ตำแหน่งนั้น พลิกตำแหน่งที่ 5 จาก 1 เป็น 0 ⟹ ได้ **0110011** ซึ่งตรงกับรหัสที่เราเข้ารหัสไว้ในข้อ Y7.1 พอดี ✓

ขั้น 5 — ดึงข้อมูลกลับ ทิ้งตำแหน่ง 1, 2, 4 (parity) เก็บตำแหน่ง 3, 5, 6, 7 ⟹ ข้อมูลเดิมคือ **1011** ✓

ตอบ: error อยู่ที่ตำแหน่งที่ 5 ⟹ รหัสที่ถูกต้องคือ 0110011 และข้อมูลคือ 1011

“Checks C1,C4 fail, so the syndrome is 5 and bit 5 must be flipped.”

STOP

หน้าเฉลย Assignment #2

ตั้งแต่หน้านี้ไปคือเฉลยข้อ 4 ถึงข้อ 10 ของ Assignment #2 ครับ ถ้าน้องยังไม่ได้ลองทำเอง พี่ขอให้กลับไปทำก่อนนะครับ การอ่านเฉลยแล้วรู้สึกว่า “อ๋อ ง่ายจัง” มันคนละเรื่องกับการทำเองได้ในห้องสอบเลยครับ

8 เฉลย Assignment ข้อ 4–6 (BCD และ Gray code)

ทุกข้อย่อยพี่ทำครบ 6 ชั้นเหมือนกันหมดครับ — โจทย์ต้นฉบับ · เตรียมก่อนลงมือ · ไล่ทีละขั้น · คำตอบ · ตรวจด้วยวิธีที่สอง · กับดักของข้อนั้น

ข้อ 4 — Convert the following numbers using binary-coded decimals

ข้อ 4 (a)

4 (a) Assignment #2 · ข้อ 4a

> $011100001001.0010_{\text{BCD}} \Rightarrow$ decimal numbers

แปล: จงแปลงรหัส BCD นี้กลับเป็นเลขฐานสิบครับ

ขั้นที่ 1 · โจทย์ซ่อนอะไรไว้

1. ตัวห้อยเขียนว่า BCD ไม่ใช่ 2 — ห้ามคิดเป็นเลขฐานสองก้อนเดียวเด็ดขาดครับ

2. มีจุดทศนิยม $\Rightarrow$ ต้อง**ยืนที่จุดแล้วหันออกไปสองทาง** ไม่ใช่หันจากปลายซ้ายหรือปลายขวารวด

3. นับบิตให้ดี — ฝั่งซ้ายจุดมี 12 บิต (ได้ 3 หลัก) ฝั่งขวาจุดมี 4 บิต (ได้ 1 หลัก) ถ้าหันผิดจะได้จำนวนหลักไม่ตรง

ชั้นที่ 2 · เตรียมก่อนลงมือ — ใช้กฎเดียวคือ “หนึ่งหลักฐานสิบ = 4 บิต” และอ่านแต่ละก้อนด้วยน้ำหนัก **8-4-2-1** ครับ เลือกวิธีนี้เพราะเป็นนิยามของ BCD โดยตรง ไม่มีการคำนวณอื่น

ชั้นที่ 3 · ไล่ทีละขั้น

ขั้น 1 — ยืนที่จุดทศนิยม แล้วหั่นออกไปสองทาง ทางละ 4 บิต

```
0111 0000 1001 . 0010
 ก1   ก2   ก3    ก4
```

ก้อนที่	บิต 4 ตัว	คิดด้วยน้ำหนัก 8-4-2-1	ได้หลัก	อยู่ในช่วง 0000–1001 ไหม
ก1	0111	4 + 2 + 1	**7**	ผ่าน
ก2	0000	ไม่มีบิตติดเลย	**0**	ผ่าน
ก3	1001	8 + 1	**9**	ผ่าน
ก4	0010	2	**2**	ผ่าน

ขั้น 2 — เรียงกลับตามลำดับเดิม โดยวางจุดไว้ที่เดิม เพราะตำแหน่งจุดไม่ได้เปลี่ยนไปไหน: ก1 ก2 ก3 แล้วจุด แล้ว ก4 ⟹ 7, 0, 9 . 2

ชั้นที่ 4 · คำตอบข้อ 4 (a)

$$011100001001.0010_{\text{BCD}} = \mathbf{709.2_{10}}$$

“Splitting the code into four-bit groups from the point gives 0111 0000 1001 . 0010, that is 709.2 in decimal.”

ชั้นที่ 5 · ตรวจด้วยวิธีที่สอง

เดินย้อนทาง — เอา 709.2 แปลงกลับเป็น BCD: 7 ⟹ 0111 · 0 ⟹ 0000 · 9 ⟹ 1001 · จุด · 2 ⟹ 0010 ⟹ ได้ 011100001001.0010 ตรงกับโจทย์ทุกบิต ✓

ตรวจด้วยเครื่องมือ `from_bcd("011100001001")` ได้ **709** และ `from_bcd("0010")` ได้ **2** ✓

ชั้นที่ 6 · กับดักของข้อ 4 (a)

คำตอบผิดที่จะได้ถ้าพลาด: ถ้าเผลออ่านทั้งก้อนเป็นเลขฐานสองปกติ $011100001001_2 = 1801_{10}$ ซึ่ง**ผิดสนิท**ครับ ห่างจาก 709 คนละเรื่อง

อีกอันหนึ่ง: ถ้าหั่นจากซ้ายสุดโดยไม่สนใจจุด จะได้ก้อนสุดท้ายไม่ครบ 4 บิต แล้วเพี้ยนทั้งแถวครับ

ข้อ 4 (b)

4 (b) Assignment #2 · ข้อ 4b

> $8013_{10} \Rightarrow$ BCD

แปล: จงแปลง 8013 ฐานสิบ เป็นรหัส BCD ครับ

ชั้นที่ 1 · โจทย์ซ่อนอะไรไว้

1. ขาไปคราวนี้ ไม่ใช่ขากลับ — โจทย์ให้ฐานสิบมาแล้วขอ BCD ครับ

2. มีเลข 0 อยู่กลางเลข — คนชอบข้ามเลข 0 ไป แต่**เลข 0 ก็เป็นหนึ่งหลัก ต้องเขียน 0000 ให้ครบ 4 บิต เหมือนกัน**

3. ทำนายความยาวไว้ก่อน — 4 หลัก × 4 บิต = **16 บิต** ไว้ตรวจตอนจบครับ

ชั้นที่ 2 · เตรียมก่อนลงมือ — หั่นเป็นหลักก่อน แล้วแปลงทีละหลักด้วยน้ำหนัก 8-4-2-1 เลือกวิธีนี้เพราะ BCD ทำงานทีละหลักตามนิยาม **ห้ามเอา 8013 ไปหารสั้นด้วย 2 แบบ EP01 เด็ดขาด**

ชั้นที่ 3 · ไล่ทีละขั้น

ขั้น 1 — หั่นเป็นหลัก

`8  0  1  3`

หลัก	เลือกน้ำหนักที่บวกได้เท่านี้	รหัส 4 บิต	เหตุผล
8	8	**1000**	ติดแค่บิตน้ำหนัก 8
0	ไม่ต้องติดบิตไหนเลย	**0000**	ห้ามข้าม ต้องเขียน 4 บิตให้ครบ
1	1	**0001**	ติดแค่บิตน้ำหนัก 1
3	2 + 1	**0011**	ติดบิตน้ำหนัก 2 กับ 1

ขั้น 2 — ต่อกันตามลำดับหลักเดิม เพราะสลับที่ไม่ได้ (หลักพันต้องอยู่ซ้ายสุด)

ชั้นที่ 4 · คำตอบข้อ 4 (b)

$$8013_{10} = \mathbf{1000\ 0000\ 0001\ 0011}_{\text{BCD}}$$

นับได้ 16 บิตพอดีตามที่ทำนายไว้ ✓

“Each decimal digit is encoded in its own four bits, so 8013 becomes 1000 0000 0001 0011.”

ขั้นที่ 5 · ตรวจด้วยวิธีที่สอง

เดินย้อนทาง — หั่น 1000|0000|0001|0011 แล้วอ่าน: 8, 0, 1, 3 ⟹ 8013 ตรงกับโจทย์ ✓ และไม่มีก้อนไหนตกในช่วงต้องห้าม 1010–1111 เลย ✓

ตรวจด้วยเครื่องมือ `to_bcd(8013)` ได้ **"1000 0000 0001 0011"** ✓

ขั้นที่ 6 · กับดักของข้อ 4 (b)

คำตอบผิดที่จะได้ถ้าพลาด: ถ้าเผลอแปลง 8013 เป็นเลขฐานสองปกติจะได้ 1111101001101_2 (13 บิต) ซึ่ง**ไม่ใช่ BCD** — สังเกตง่าย ๆ ว่า**ความยาวไม่หารด้วย 4 ลงตัว** นั่นคือสัญญาณเตือนว่าทำผิดวิธีครับ

อีกอันหนึ่ง: ถ้าข้ามเลข 0 ไปจะได้ 1000 0001 0011 (12 บิต) แล้วอ่านกลับได้ 813 ไม่ใช่ 8013 ครับ

ข้อ 5 — Convert the following binary number to Gray code

ขั้นที่ 1–2 · ใช้ร่วมกันทั้ง (a) และ (b)

โจทย์ซ่อนอะไร: ตัวห้อยเป็น $_2$ แล้วขอ Gray ⟹ นี่คือ**ขาไป**ครับ ⟹ ใช้กฎ “MSB ลอกมา แล้ว XOR ทุกคู่ที่ติดกันของ**เลขฐานสองต้นฉบับ**” (ทุกคู่หยิบจากแถวบนล้วน ไม่มีการทบผลลัพธ์)

เตรียมก่อนลงมือ: ทั้งสองข้อให้มา 8 บิต ⟹ ต้องตอบ **8 บิต** และจะมีการ XOR ทั้งหมด **7 คู่** (8 บิตวางคู่ติดกันได้ 7 คู่) ⟹ ใช้ตัวเลขนี้ตรวจว่าทำครบไหมครับ

ข้อ 5 (a)

5 (a) Assignment #2 · ข้อ 5a

> $01000101_2 \Rightarrow$ Gray

ขั้นที่ 3 · ไล่ทีละขั้น — ขั้น 1 ลอก MSB ลงมาได้ **0** · ขั้น 2 XOR ทีละคู่จากซ้ายไปขวา:

คู่ที่	บิตคู่ที่ติดกัน (จากต้นฉบับ)	ผล XOR	ได้ Gray บิตที่
—	ลอก MSB มาตรง ๆ	**0**	1
1	$0 \oplus 1$	**1**	2
2	$1 \oplus 0$	**1**	3
3	$0 \oplus 0$	**0**	4
4	$0 \oplus 0$	**0**	5

คู่ที่	บิตคู่ที่ติดกัน (จากต้นฉบับ)	ผล XOR	ได้ Gray บิตที่
5	$0 \oplus 1$	**1**	6
6	$1 \oplus 0$	**1**	7
7	$0 \oplus 1$	**1**	8

เรียงผลลัพธ์ทั้ง 8 บิต

```
Binary   0 1 0 0 0 1 0 1
Gray     0 1 1 0 0 1 1 1
```

ขั้นที่ 4 · คำตอบข้อ 5 (a)

$01000101_2 = \mathbf{01100111}_{\text{Gray}}$

“Copy the MSB, then XOR each adjacent pair: 0100 0101 becomes 0110 0111 in Gray code.”

ขั้นที่ 5 · ตรวจด้วยวิธีที่สอง

แปลงกลับ — เอา 01100111 ทำขากลับ (ทบต่อกัน): $b_1 = 0 \cdot 0 \oplus 1 = 1 \cdot 1 \oplus 1 = 0 \cdot 0 \oplus 0 = 0 \cdot 0 \oplus 0 = 0 \cdot 0 \oplus 1 = 1 \cdot 1 \oplus 1 = 0 \cdot 0 \oplus 1 = 1 \Longrightarrow$ ได้ 01000101 ตรงกับโจทย์ ✓

ตรวจด้วยเครื่องมือ `to_gray("01000101")` ได้ **"01100111"** ✓

ขั้นที่ 6 · กับดักของข้อ 5 (a)

คำตอบผิดที่จะได้ถ้าพลาด: ถ้าเผลอทำแบบขากลับ (เอาผลลัพธ์ที่เพิ่งได้ไปทบกับบิตถัดไป) จะได้ **01111001** ซึ่งผิดครับ — สังเกตว่ามันเริ่มเพี้ยนตั้งแต่บิตที่ 3 เป็นต้นไป

อีกอันหนึ่ง: ถ้าตัด 0 นำหน้าทิ้งแล้วตอบ 1100111 (7 บิต) ก็ถือว่าผิดรูปแบบครับ ต้องตอบ 8 บิตเท่าโจทย์

ข้อ 5 (b)

5 (b) Assignment #2 · ข้อ 5b

> $01110110_2 \Rightarrow$ Gray

ขั้นที่ 3 · ไล่ทีละขั้น — ขั้น 1 ลอก MSB ได้ **0** · ขั้น 2 XOR ทีละคู่ (ทำครบ 7 คู่เหมือนเดิมครับ ไม่ลัด):

คู่ที่	บิตคู่ที่ติดกัน (จากต้นฉบับ)	ผล XOR	ได้ Gray บิตที่
—	ลอก MSB มาตรง ๆ	**0**	1

คู่ที่	บิตคู่ที่ติดกัน (จากต้นฉบับ)	ผล XOR	ได้ Gray บิตที่
1	$0 \oplus 1$	**1**	2
2	$1 \oplus 1$	**0**	3
3	$1 \oplus 1$	**0**	4
4	$1 \oplus 0$	**1**	5
5	$0 \oplus 1$	**1**	6
6	$1 \oplus 1$	**0**	7
7	$1 \oplus 0$	**1**	8

เรียงผลลัพธ์ทั้ง 8 บิต

```
Binary   0 1 1 1 0 1 1 0
Gray     0 1 0 0 1 1 0 1
```

ชั้นที่ 4 · คำตอบข้อ 5 (b)

$01110110_2 = \mathbf{01001101}_{\text{Gray}}$

“0111 0110 in binary is 0100 1101 in Gray code.”

ชั้นที่ 5 · ตรวจด้วยวิธีที่สอง

แปลงกลับ — เอา 01001101 ทบต่อกัน: $b_1 = 0 \cdot 0 \oplus 1 = 1 \cdot 1 \oplus 0 = 1 \cdot 1 \oplus 0 = 1 \cdot 1 \oplus 1 = 0 \cdot 0 \oplus 1 = 1 \cdot 1 \oplus 0 = 1 \cdot 1 \oplus 1 = 0 \Longrightarrow$ ได้ 01110110 ตรงกับโจทย์ ✓

ตรวจด้วยเครื่องมือ `to_gray("01110110")` ได้ **"01001101"** ✓

ชั้นที่ 6 · กับดักของข้อ 5 (b)

ระวังตรงกลาง — ข้อนี้มี 1 ติดกันสามตัว (111) ทำให้เกิด $1 \oplus 1 = 0$ ติดกันสองครั้ง คนมักเผลอมองว่า “มี 1 อยู่ก็ตอบ 1” แล้วเขียนผิดตรงคู่ที่ 2 และคู่ที่ 3 ครับ

จำไว้ว่า **เหมือนกันได้ 0 · ต่างกันได้ 1** — ตัดสินจาก “เหมือนหรือต่าง” ไม่ใช่จากค่าของบิต

ข้อ 6 — Convert the following Gray code to binary numbers

ขั้นที่ 1–2 · ใช้ร่วมกันทั้ง (a) และ (b)

โจทย์ซ่อนอะไร: ตัวห้อยเป็น $_{\text{Gray}}$ แล้วขอ base 2 ⟹ นี่คือ**ขากลับ** ⟹ กฎเปลี่ยนเป็น “MSB ลอกมา แล้วเอา**ผลลัพธ์ตัวที่เพิ่งได้** XOR กับ Gray บิตถัดไป”

สิ่งที่ต่างจากข้อ 5 มาก: ขากลับ**ข้ามไปทำบิตขวาสุดก่อนไม่ได้** เพราะทุกก้าวต้องรอผลของก้าวก่อนหน้า ⟹ ต้องเดินจากซ้ายไปขวาทีละก้าวเท่านั้นครับ

เตรียมก่อนลงมือ: ให้มา 8 บิต ⟹ ตอบ 8 บิต และจะมี 7 ก้าวหลังจากลอก MSB

ข้อ 6 (a)

6 (a) Assignment #2 · ข้อ 6a

> $01101011_{\text{Gray}} \Rightarrow$ base 2

ขั้นที่ 3 · ไล่ทีละขั้น — ขั้น 1 ลอก MSB ได้ $b_1 = \mathbf{0}$ · ขั้น 2 ทบต่อกันทีละก้าว:

ก้าวที่	สูตร (ผลตัวก่อน XOR Gray ตัวถัดไป)	แทนค่า	ได้ binary บิตที่
—	$b_1 = g_1$ (ลอกมา)	$= \mathbf{0}$	1
1	$b_2 = b_1 \oplus g_2$	$0 \oplus 1 = \mathbf{1}$	2
2	$b_3 = b_2 \oplus g_3$	$1 \oplus 1 = \mathbf{0}$	3
3	$b_4 = b_3 \oplus g_4$	$0 \oplus 0 = \mathbf{0}$	4
4	$b_5 = b_4 \oplus g_5$	$0 \oplus 1 = \mathbf{1}$	5
5	$b_6 = b_5 \oplus g_6$	$1 \oplus 0 = \mathbf{1}$	6
6	$b_7 = b_6 \oplus g_7$	$1 \oplus 1 = \mathbf{0}$	7
7	$b_8 = b_7 \oplus g_8$	$0 \oplus 1 = \mathbf{1}$	8

เรียงผลลัพธ์ทั้ง 8 บิต

```
Gray      0 1 1 0 1 0 1 1
Binary    0 1 0 0 1 1 0 1
```

ขั้นที่ 4 · คำตอบข้อ 6 (a)

$01101011_{\text{Gray}} = \mathbf{01001101_2}$ (มีค่าเท่ากับ $64 + 8 + 4 + 1 = \mathbf{77_{10}}$)

“Gray 0110 1011 converts to binary 0100 1101, which is 77 in decimal.”

ชั้นที่ 5 · ตรวจด้วยวิธีที่สอง

แปลงไปใหม่ — เอา 01001101 ทำขาไป (XOR คู่ติดกันจากแถวบนล้วน): $0 \cdot 0 \oplus 1 = 1 \cdot 1 \oplus 0 = 1 \cdot 0 \oplus 0 = 0 \cdot 0 \oplus 1 = 1 \cdot 1 \oplus 1 = 0 \cdot 1 \oplus 0 = 1 \cdot 0 \oplus 1 = 1 \Longrightarrow$ ได้ 01101011 ตรงกับโจทย์ ✓

ตรวจด้วยเครื่องมือ `from_gray("01101011")` ได้ **"01001101"** ✓

ชั้นที่ 6 · กับดักของข้อ 6 (a)

คำตอบผิดที่จะได้ถ้าพลาด: ถ้าเผลอใช้สูตรขาไป (XOR คู่ติดกันของ Gray เอง) จะได้ **01011110** ซึ่งผิดครับ

และ**ห้ามคิดค่าฐานสิบจาก Gray ตรง ๆ** — ถ้าเอา 01101011 ไปคูณน้ำหนักจะได้ 107 แต่คำตอบที่ถูกคือ **77** ครับ ต้องแปลงเป็นฐานสองก่อนเสมอ

ข้อ 6 (b)

6 (b) Assignment #2 · ข้อ 6b

> $11011111_{\text{Gray}} \Rightarrow$ base 2

ชั้นที่ 3 · ไล่ทีละขั้น — ขั้น 1 ลอก MSB ได้ $b_1 = \mathbf{1}$ · ขั้น 2 ทบต่อกันทีละก้าว:

ก้าวที่	สูตร (ผลตัวก่อน XOR Gray ตัวถัดไป)	แทนค่า	ได้ binary บิตที่
—	$b_1 = g_1$ (ลอกมา)	$= \mathbf{1}$	1
1	$b_2 = b_1 \oplus g_2$	$1 \oplus 1 = \mathbf{0}$	2
2	$b_3 = b_2 \oplus g_3$	$0 \oplus 0 = \mathbf{0}$	3
3	$b_4 = b_3 \oplus g_4$	$0 \oplus 1 = \mathbf{1}$	4
4	$b_5 = b_4 \oplus g_5$	$1 \oplus 1 = \mathbf{0}$	5
5	$b_6 = b_5 \oplus g_6$	$0 \oplus 1 = \mathbf{1}$	6
6	$b_7 = b_6 \oplus g_7$	$1 \oplus 1 = \mathbf{0}$	7
7	$b_8 = b_7 \oplus g_8$	$0 \oplus 1 = \mathbf{1}$	8

เรียงผลลัพธ์ทั้ง 8 บิต

```
Gray     1 1 0 1 1 1 1 1
Binary   1 0 0 1 0 1 0 1
```

ชั้นที่ 4 · คำตอบข้อ 6 (b)

$11011111_{\text{Gray}} = \mathbf{10010101_2}$ (มีค่าเท่ากับ $128 + 16 + 4 + 1 = \mathbf{149_{10}}$)

“Gray 1101 1111 converts to binary 1001 0101, which is 149 in decimal.”

ชั้นที่ 5 · ตรวจด้วยวิธีที่สอง

แปลงไปใหม่ — เอา 10010101 ทำขาไป: $1 \cdot 1 \oplus 0 = 1 \cdot 0 \oplus 0 = 0 \cdot 0 \oplus 1 = 1 \cdot 1 \oplus 0 = 1 \cdot 0 \oplus 1 = 1 \cdot 1 \oplus 0 = 1 \cdot 0 \oplus 1 = 1 \Longrightarrow$ ได้ 11011111 ตรงกับโจทย์ ✓

ตรวจด้วยเครื่องมือ `from_gray("11011111")` ได้ **"10010101"** ✓

ชั้นที่ 6 · กับดักของข้อ 6 (b)

ข้อนี้มี 1 ติดกันห้าตัวท้าย ๆ ทำให้ผลลัพธ์สลับ 0-1 ไปมา คนมักเผลอเขียน 1 รวดตามที่ตาเห็น ทั้งที่ผลจริงต้องสลับ 0 กับ 1 ไปเรื่อย ๆ ครับ

เทคนิคกันพลาด: **เขียนผลลัพธ์ทีละตัวใต้ Gray แต่ละบิตเลย** แล้วชี้นิ้วไล่ทีละคู่ อย่าคิดในใจรวดเดียวครับ

หัวใจของข้อ 4–6: อ่านตัวห้อยให้ออกก่อนเสมอ — $_{\text{BCD}}$ หั่นทีละ 4 บิต · $_2$ ขอ Gray คือขาไป · $_{\text{Gray}}$ ขอ base 2 คือขากลับ

9 เฉลย Assignment ข้อ 7–8 (Parity)

อ่านก่อนเริ่ม — สองข้อนี้ต่างกันตรงไหน

ข้อ 7 เป็นงานฝั่งส่ง — โจทย์บอกว่า “assign the proper **even** parity bit” คือให้เรา**สร้าง** parity bit ขึ้นมาเติมให้ครับ

ข้อ 8 เป็นงานฝั่งรับ — โจทย์บอกว่า “an **odd** parity system **receives**” คือของที่ให้มา**มี parity bit ติดมาแล้ว** หน้าที่เราคือ**ตรวจ** ไม่ใช่สร้าง

และคนละ parity กันด้วย! ข้อ 7 เป็น **even** ข้อ 8 เป็น **odd** ถ้าอ่านข้ามแล้วใช้กฎเดียวกันทั้งสองข้อ จะผิดหมดครับ

ข้อ 7 — Assign the proper even parity bit to the following code groups

ชั้นที่ 2 · เตรียมก่อนลงมือ (ใช้ร่วมกันทั้ง a และ b) — ใช้กฎ 2 ขั้นคือ **“นับจำนวน 1 ก่อน แล้วเติมบิตที่ทำให้ผลรวมเป็นคู่”** ครับ เลือกวิธีนี้เพราะ even parity นิยามไว้แบบนั้นตรง ๆ ไม่มีการคำนวณอย่างอื่นเลย

กฎที่จะใช้ทั้งสองข้อ (even parity)

```
นับ 1 ได้จำนวนคู่   ->  เติม 0   (ไม่ต้องเปลี่ยนอะไร)
นับ 1 ได้จำนวนคี่   ->  เติม 1   (เพิ่มอีก 1 ตัวให้กลายเป็นคู่)
```

ข้อ 7 (a)

7 (a) — Assignment #2 · ข้อ 7a

> 101101_2

แปล: จงเติม parity bit แบบคู่ให้กับรหัสชุดนี้ครับ

ชั้นที่ 1 · โจทย์ซ่อนอะไรไว้

1. คำว่า even — ต้องทำให้ผลรวมเป็นคู่ ถ้าอ่านเป็น odd คำตอบจะพลิกทันที

2. ข้อมูลยาว 6 บิต ⟹ เติม parity แล้วต้องได้ **7 บิต** เสมอ ไว้ตรวจตอนจบครับ

3. โจทย์ไม่ได้บอกว่าให้วางหัวหรือท้าย — สไลด์วางไว้**หน้าสุด** พี่จะวางหน้าสุดตามสไลด์ครับ แต่วางท้ายก็ไม่ผิด ขอแค่**เขียนกำกับให้ชัดว่าตัวไหนคือ parity bit**

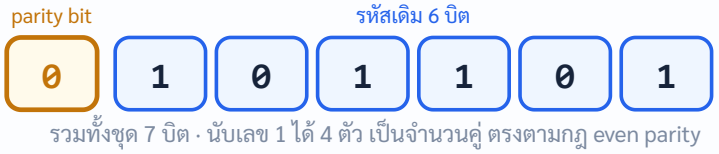
ชั้นที่ 3 · ไล่ทีละขั้น

ตำแหน่งที่	1	2	3	4	5	6	นับสะสม
บิต	1	0	1	1	0	1	—
นับ 1 ได้กี่ตัวแล้ว	1	1	2	3	3	**4**	**รวม 4 ตัว**

ขั้น 1 — นับจำนวน 1 ไล่ทีละตำแหน่งตามตารางข้างบน ได้ **4 ตัว** (ช่องเขียวคือตำแหน่งที่เป็น 1 ครับ)

ขั้น 2 — เทียบกับเป้าหมาย 4 เป็นจำนวน**คู่**อยู่แล้ว ตรงกับที่ even parity ต้องการ ⟹ **ไม่ต้องเพิ่มเลข 1 เข้าไปอีก** ⟹ parity bit = **0**

ขั้น 3 — เขียนชุดเต็ม วาง parity ไว้หน้าสุดตามสไลด์



รวมทั้งชุด 7 บิต · นับเลข 1 ได้ 4 ตัว เป็นจำนวนคู่ ตรงตามกฎ even parity

แบบเดียวกับที่สไลด์หน้า 40 วางไว้ — ขีดเส้นใต้ตัวหน้าสุดคือ parity bit

ชั้นที่ 4 · คำตอบข้อ 7 (a)

even parity bit = 0 ⟹ ชุดเต็มคือ 0 101101

“The group 101101 already contains four 1s, an even number, so the even parity bit is 0 and the transmitted group is 0101101.”

ชั้นที่ 5 · ตรวจด้วยวิธีที่สอง

ตรวจแบบฝั่งรับ — เอาชุดเต็ม 0101101 มานับ 1 ใหม่ทั้งชุด ได้ 4 ตัว เป็นคู่ ✓ ผ่านกฎ even parity แปลว่า parity bit ที่เติมไปถูกต้อง

ตรวจด้วยเครื่องมือ `parity_bit("101101", even=True)` ได้ **"0"** และ `parity_ok("0101101", even=True)` ได้ **True** ✓

ชั้นที่ 6 · กับดักของข้อ 7 (a)

คำตอบผิดที่จะได้ถ้าพลาด: ถ้าเผลออ่านว่าเป็น odd parity จะตอบ parity bit = **1** (ชุดเต็ม 1101101) ซึ่งผิดโจทย์ครับ — **วงกลมคำว่า even/odd ในโจทย์ทุกครั้งก่อนลงมือ**

อีกอันหนึ่ง: **อย่านับผิดเพราะบิต 0 ปนอยู่** — parity สนใจแค่จำนวนเลข 1 เลข 0 มีกี่ตัวก็ไม่มีผลเลยครับ

ข้อ 7 (b)

7 (b) Assignment #2 · ข้อ 7b

> 1000111001001_2

ชุดนี้ยาว 13 บิต ต้องนับให้ดี ๆ ครับ

ขั้นที่ 1 · โจทย์ซ่อนอะไรไว้

1. ยาว 13 บิต — เป็นเลขคี่ ทำให้นับพลาดง่ายมาก **ต้องจับกลุ่มก่อนนับ** อย่านับรวดเดียว

2. ยังเป็น even เหมือนข้อ (a) — โจทย์ข้อ 7 บอก even ครอบทั้งสองข้อย่อยครับ

3. ชุดเต็มต้องได้ 14 บิต ไว้ตรวจตอนจบ

ขั้นที่ 3 · ไล่ทีละขั้น

ขั้น 1 — จับกลุ่มก่อนนับ เพราะ 13 บิตเรียงยาวทำให้ตาลาย พี่แบ่งเป็นกลุ่มละ 4 บิตจากซ้าย เหลือกลุ่มสุดท้าย 1 บิตครับ

แบ่งกลุ่มแล้วนับทีละกลุ่ม

```
1000  1110  0100  1
  1     3     1    1
-------------------
รวม = 1 + 3 + 1 + 1 = 6 ตัว
```

กลุ่ม	บิต	มีเลข 1 กี่ตัว	ตำแหน่งของ 1 (นับจากซ้ายทั้งชุด)
กลุ่มที่ 1	1000	**1**	ตำแหน่ง 1
กลุ่มที่ 2	1110	**3**	ตำแหน่ง 5, 6, 7
กลุ่มที่ 3	0100	**1**	ตำแหน่ง 10
กลุ่มที่ 4	1	**1**	ตำแหน่ง 13
รวม	13 บิต	**6 ตัว**	1, 5, 6, 7, 10, 13

ขั้น 2 — เทียบกับเป้าหมาย 6 เป็นจำนวน**คู่**อยู่แล้ว ตรงกับ even parity ⟹ **parity bit = 0**

ขั้น 3 — เขียนชุดเต็ม วาง parity ไว้หน้าสุด ⟹ ได้ 14 บิตพอดีตามที่ทำนายไว้ ✓

ขั้นที่ 4 · คำตอบข้อ 7 (b)

even parity bit = 0 ⟹ ชุดเต็มคือ 0 1000111001001

“The group contains six 1s, an even number, so the even parity bit is 0.”

ชั้นที่ 5 · ตรวจด้วยวิธีที่สอง

นับใหม่คนละทาง — คราวนี้นับ**จากขวาไปซ้าย**ครับ (นับคนละทิศเป็นการตรวจที่ดี เพราะถ้านับพลาดจะไม่พลาดซ้ำรอยเดิม): ตำแหน่งที่เป็น 1 คือ 13, 10, 7, 6, 5, 1 ⟹ ยัง 6 ตัวเท่าเดิม ✓

ตรวจแบบฝั่งรับ — ชุดเต็ม 01000111001001 นับ 1 ได้ 6 ตัว เป็นคู่ ✓

ตรวจด้วยเครื่องมือ `parity_bit("1000111001001", even=True)` ได้ **"0"** และ `parity_ok("01000111001001", even=True)` ได้ **True** ✓

ชั้นที่ 6 · กับดักของข้อ 7 (b)

คำตอบผิดที่จะได้ถ้าพลาด: ถ้านับพลาดไปตัวเดียว เช่นได้ 5 หรือ 7 จะตอบ parity bit = **1** ซึ่งผิดครับ — **บิตยาว ๆ ต้องจับกลุ่มก่อนนับเสมอ**

ข้อนี้มี 1 ติดกันสามตัวตรงกลาง (111) ซึ่งเป็นจุดที่คนนับผิดบ่อยที่สุด บางคนนับเป็น 2 บ้าง 4 บ้าง — **ชี้นิ้วไล่ทีละตัวครับ อย่ากวาดตา**

ข้อ 8 — An odd parity system receives the following code groups. Determine which groups (if any) contain a single bit error

ชั้นที่ 2 · เตรียมก่อนลงมือ (ใช้ร่วมกันทั้ง a และ b) — คราวนี้เป็นงานฝั่งรับ ใช้กฎเดียวคือ **“นับ 1 ทั้งชุด (รวม parity bit ที่ติดมาแล้ว) แล้วดูว่าเป็นคี่ไหม”** ครับ

กฎที่จะใช้ทั้งสองข้อ (odd parity system รับข้อมูล)

```
นับ 1 ได้จำนวนคี่   ->  ตรงตามกฎ   ->  ไม่พบ error
นับ 1 ได้จำนวนคู่   ->  ผิดกฎ      ->  พบ error (มีบิตพลิกจำนวนคี่)
```

ชั้นที่ 1 · โจทย์ซ่อนอะไรไว้ (ทั้ง a และ b)

1. คำว่า “receives” — ของที่ให้มา**มี parity bit ติดมาแล้ว** ห้ามเติมอะไรเข้าไปอีก และห้ามแยก parity bit ออกก่อนนับครับ

2. คำว่า “odd” — ข้อ 7 เป็น even แต่ข้อนี้เป็น **odd** เกณฑ์กลับด้านกันเลย

3. คำว่า “if any” — โจทย์เปิดช่องไว้ว่าอาจไม่มีข้อไหนผิดก็ได้ เพราะฉะนั้นอย่าเดาว่าต้องมีข้อผิดแน่ ๆ ต้องตรวจจริงทั้งสองข้อครับ

4. คำว่า “single bit error” — สำคัญมาก เพราะ parity จับได้เฉพาะบิตพลิก**จำนวนคี่** ถ้าเจอชุดที่ผ่าน ต้องตอบว่า “ไม่พบ error” ไม่ใช่ “ไม่มี error” ครับ

ข้อ 8 (a)

8 (a) Assignment #2 · ข้อ 8a

> 110011_2

ชั้นที่ 3 · ไล่ทีละขั้น

ตำแหน่งที่	1	2	3	4	5	6
บิต	1	1	0	0	1	1
นับสะสม	1	2	2	2	3	**4**

ขั้น 1 — นับ 1 ทั้งชุด ได้ **4 ตัว** (ตำแหน่ง 1, 2, 5, 6)

ขั้น 2 — เทียบกับกฎของระบบ ระบบเป็น **odd** parity ⟹ ต้องได้จำนวน**คี่** แต่นับได้ 4 ซึ่งเป็น**คู่** ⟹ **ขัดกับกฎ**

ขั้น 3 — สรุป การที่ผลออกมาเป็นคู่ทั้งที่ควรเป็นคี่ แปลว่ามีบิตพลิกไป**จำนวนคี่ตัว** (1 ตัว หรือ 3 ตัว หรือ 5 ตัว) ซึ่งกรณีที่โจทย์ถามคือ **single bit error** ⟹ **ชุดนี้มี error**

ชั้นที่ 4 · คำตอบข้อ 8 (a)

110011_2 มีเลข 1 อยู่ 4 ตัว ซึ่งเป็นจำนวนคู่ ⟹ ผิดกฎ odd parity ⟹ ชุดนี้ มี error

“The group 110011 contains four 1s, an even number. In an odd parity system this violates the rule, so this group contains a single bit error.”

ชั้นที่ 5 · ตรวจด้วยวิธีที่สอง

คิดกลับทาง — ถ้าชุดนี้ถูกต้องในระบบ odd parity มันควรมีเลข 1 เป็นจำนวนคี่ เช่น 3 หรือ 5 ตัว แต่มันมี 4 ตัว ⟹ ห่างจากค่าที่ถูกต้องอยู่ 1 ตัวพอดี ⟹ สอดคล้องกับการพลิก 1 บิต ✓

ตรวจด้วยเครื่องมือ `parity_ok("110011", even=False)` ได้ **False** ⟹ ไม่ผ่านกฎ odd parity ⟹ มี error ✓

ชั้นที่ 6 · กับดักของข้อ 8 (a)

คำตอบผิดที่จะได้ถ้าพลาด: ถ้าเผลอใช้เกณฑ์ **even** (ติดมาจากข้อ 7) จะเห็นว่า 4 เป็นคู่แล้วตอบว่า “ผ่าน ไม่มี error” ซึ่ง**ผิดตรงกันข้ามกับคำตอบจริงเลย**ครับ

อีกอันหนึ่ง: **ห้ามตอบว่าผิดที่บิตไหน** — parity บอกได้แค่ว่ามี error ถ้าเขียนว่า “ผิดที่บิตที่ 3” จะโดนหักคะแนน เพราะไม่มีข้อมูลรองรับเลยครับ

ข้อ 8 (b)

8 (b) Assignment #2 · ข้อ 8b

> 1100010101010_2

ยาว 13 บิตอีกแล้วครับ จับกลุ่มก่อนนับ

ชั้นที่ 3 · ไล่ทีละขั้น

แบ่งกลุ่มแล้วนับทีละกลุ่ม

```
1100   0101   0101   0
  2      2      2     0
-------------------------
รวม = 2 + 2 + 2 + 0 = 6 ตัว
```

กลุ่ม	บิต	มีเลข 1 กี่ตัว	ตำแหน่งของ 1 (นับจากซ้ายทั้งชุด)
กลุ่มที่ 1	`1100`	**2**	ตำแหน่ง 1, 2
กลุ่มที่ 2	`0101`	**2**	ตำแหน่ง 6, 8
กลุ่มที่ 3	`0101`	**2**	ตำแหน่ง 10, 12
กลุ่มที่ 4	`0`	**0**	—
รวม	13 บิต	**6 ตัว**	1, 2, 6, 8, 10, 12

ขั้น 1 — นับ 1 ทั้งชุด ได้ **6 ตัว**

ขั้น 2 — เทียบกับกฎของระบบ ระบบเป็น **odd** ⟹ ต้องได้จำนวนคี่ แต่ได้ 6 ซึ่งเป็นคู่ ⟹ **ขัดกับกฎ**

ขั้น 3 — สรุป ⟹ ชุดนี้มี error เหมือนกัน ครับ

ชั้นที่ 4 · คำตอบข้อ 8 (b) และสรุปทั้งข้อ 8

1100010101010_2 **มีเลข 1 อยู่ 6 ตัว ซึ่งเป็นจำนวนคู่ ⟹ ผิดกฎ odd parity ⟹ ชุดนี้ มี error**

สรุปข้อ 8: ทั้งสองชุดมี error ครับ — (a) มี 1 อยู่ 4 ตัว และ (b) มี 1 อยู่ 6 ตัว ทั้งคู่เป็นจำนวนคู่ ซึ่งขัดกับ odd parity ทั้งสองชุด

“Both groups contain an even number of 1s (four and six), which violates odd parity, so both groups contain a single bit error.”

ชั้นที่ 5 · ตรวจด้วยวิธีที่สอง

นับใหม่คนละทาง — นับเฉพาะ 0 ก่อน: ชุดนี้มี 0 อยู่ 7 ตัว ⟹ จำนวน 1 = 13 − 7 = **6 ตัว** ตรงกับที่นับไว้ ✓ (เทคนิคนี้ดีเวลาชุดมี 0 น้อยกว่า 1 หรือกลับกันครับ)

ตรวจด้วยเครื่องมือ `parity_ok("1100010101010", even=False)` ได้ **False** ⟹ มี error ✓

ชั้นที่ 6 · กับดักของข้อ 8 (b)

คำตอบผิดที่จะได้ถ้าพลาด: ชุดนี้มีลาย 0101 ซ้ำ ๆ ทำให้ตาลายง่ายมาก ถ้านับได้ 5 หรือ 7 จะสรุปกลับด้านเป็น “ไม่มี error” ทันทีครับ

เทคนิคกันพลาด: ถ้าชุดมีลายซ้ำ ๆ ให้**นับตัวที่มีน้อยกว่า** แล้วเอาความยาวทั้งหมดลบออก จะแม่นกว่าเยอะครับ

และย้ำอีกครั้ง: โจทย์ถามแค่ “ชุดไหนมี error” **ไม่ได้ถามตำแหน่ง** เพราะ parity บอกตำแหน่งไม่ได้ — ถ้าอยากได้ตำแหน่งต้องใช้ Hamming ในข้อ 10 ครับ

หัวใจของข้อ 7–8: ข้อ 7 คือ “สร้าง parity ให้ครบ” (even) · ข้อ 8 คือ “ตรวจว่าที่ได้มาถูกกฎไหม” (odd) — อ่านคำว่า even/odd ให้ขาดก่อนทุกครั้ง

10 เฉลย Assignment ข้อ 9 — CRC (ข้อที่ยากที่สุดข้อหนึ่ง)

ข้อนี้กับข้อ 10 คือสองข้อที่กินเวลามากที่สุดของ Assignment นี้ครับ พี่จะไล่ให้ดู**ทุกรอบ ไม่ข้ามแม้แต่รอบเดียว** และทำครบ**ทั้งฝั่งส่งและฝั่งรับ 2 กรณี** เลย

ชั้นที่ 1 — โจทย์ต้นฉบับและสิ่งที่โจทย์ซ่อนไว้

Assignment ข้อ 9 EES271 Assignment #2 · ข้อ 9

> Determine the transmitted CRC for the following data (D) 110010 and generator code (G) 1101. Also verify that the received data 110011100 at the receiver has an error or not.

แปล: จงหา CRC ที่ต้องส่งไป เมื่อข้อมูลคือ 110010 และตัวหารคือ 1101 แล้วตรวจว่าข้อมูล 110011100 ที่ฝั่งรับได้มานั้นมี error หรือไม่ครับ

โจทย์ซ่อนอะไรไว้

1. ข้อนี้เป็นสองข้อในหนึ่งข้อ — ท่อนแรกเป็นงานฝั่งส่ง ท่อนที่สองเป็นงานฝั่งรับ ต้องทำครบทั้งคู่ถึงจะได้คะแนนเต็ม

2. ความยาวของสองท่อนไม่เท่ากัน — ข้อมูล D ยาว 6 บิต แต่ที่รับมายาว 9 บิต ซึ่งเท่ากับ 6 + 3 พอดี แปลว่า**ที่รับมาคือ "ชุดเต็มที่มี CRC ติดมาแล้ว"** $\Longrightarrow$ ฝั่งรับ**ห้ามเติมศูนย์เพิ่มอีก**ครับ

3. G ยาว 4 บิต $\Longrightarrow r = 4 - 1 = \mathbf{3} \Longrightarrow$ ฝั่งส่งเติมศูนย์ 3 ตัว และเศษที่ได้ต้องยาว 3 บิตเสมอ (สั้นกว่านั้นให้เติม 0 นำหน้า)

ชั้นที่ 2 — เตรียมก่อนลงมือ

ใช้ **4 ขั้นของฝั่งส่ง** กับ **2 ขั้นของฝั่งรับ** ตามสไลด์หน้า 43 ตรง ๆ ครับ เลือกวิธีนี้เพราะโจทย์ให้ G มาชัดเจนและถามหา "transmitted CRC" ซึ่งเป็นศัพท์ของกระบวนการนี้โดยเฉพาะ

สิ่งที่ต้องเตรียมก่อนหาร

D = **110010** (6 บิต) · G = **1101** (4 บิต) · r = **3**
ตัวตั้ง = D ต่อด้วยศูนย์ 3 ตัว = **110010000**
จำนวนรอบ = จำนวนบิตของ D = **6 รอบ**

กติกาที่ต้องท่องก่อนเริ่ม

ทุกรอบดู**บิตนำของรอบนั้น**อย่างเดียว: เป็น 1 $\Longrightarrow$ XOR ด้วย G · เป็น 0 $\Longrightarrow$ ข้าม
เป็นการ XOR ไม่ใช่การลบ — ไม่มียืม ไม่มีตัวทด ($1 \oplus 1 = 0$ เฉย ๆ ไม่ใช่ 10)

ชั้นที่ 3 — ไล่ทีละขั้น (ก) ฝั่งส่ง: หา CRC

ตัวตั้งคือ **110010000** หารด้วย **1101** ทั้งหมด **6 รอบ** ครับ ไล่ให้ดูทุกรอบ:

รอบ 1 · บิตนำ = 1 ⇒ XOR ด้วย G

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

รอบ 2 · บิตนำ = 0 ⇒ ข้าม

```
0 0 0 1 1 0 0 0 0
```

ไม่เปลี่ยน

รอบ 3 · บิตนำ = 0 ⇒ ข้าม

```
0 0 0 1 1 0 0 0 0
```

ไม่เปลี่ยน

รอบ 4 · บิตนำ = 1 ⇒ XOR ด้วย G

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

รอบ 5 · บิตนำ = 0 ⇒ ข้าม

```
0 0 0 0 0 0 1 0 0
```

ไม่เปลี่ยน

รอบ 6 · บิตนำ = 0 ⇒ ข้าม

```
0 0 0 0 0 0 1 0 0
```

ไม่เปลี่ยน

อ่านผลอย่างไร

ผลลัพธ์หลังรอบสุดท้ายคือ **000000100** ⟹ ตัด **3 บิตขวาสุด** ออกมา ได้เศษคือ **100** ⟹ นี่คือ **CRC** ที่โจทย์ถามหาครับ

แล้วขั้นที่ 4 บอกให้**เอาเศษไปแทนที่ศูนย์ที่เติมไว้** ⟹ ชุดที่ส่งออกไปจริงคือ **110010** ต่อด้วย **100** = **110010100**

ชุดที่ส่งออกไป = ข้อมูลเดิม (ฟ้า) ต่อด้วย CRC (ส้ม)



ยาว 9 บิต — เท่ากับความยาวของข้อมูลที่โจทย์ให้มาในท่อนที่สองพอดี

CRC ไม่ได้ส่งแยก แต่ไปแทนที่ศูนย์ที่จองไว้ท้ายข้อมูล

ชั้นที่ 3 — ไล่ทีละขั้น (ข) ฝั่งรับ กรณีที่ 1: ถ้าไม่มีอะไรผิดเลย

ก่อนไปตรวจของจริง พี่ขอ**ตรวจงานตัวเองก่อน**ครับ — เอาชุดที่เราเพิ่งสร้าง (110010100) ไปหารด้วย 1101 ซ้ำ ถ้าคำนวณถูก**ต้องได้เศษเป็นศูนย์ทุกบิต** (6 รอบ):

รอบ 1 · บิตนำ = 1 ⇒ XOR ด้วย G

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

รอบ 2 · บิตนำ = 0 ⇒ ข้าม

```
0 0 0 1 1 0 1 0 0
```

ไม่เปลี่ยน

รอบ 3 · บิตนำ = 0 ⇒ ข้าม

```
0 0 0 1 1 0 1 0 0
```

ไม่เปลี่ยน

รอบ 4 · บิตนำ = 1 ⇒ XOR ด้วย G

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

รอบ 5 · บิตนำ = 0 ⇒ ข้าม

```
0 0 0 0 0 0 0 0 0
```

ไม่เปลี่ยน

รอบ 6 · บิตนำ = 0 ⇒ ข้าม

```
0 0 0 0 0 0 0 0 0
```

ไม่เปลี่ยน

เศษ = 000 ⟹ เป็นศูนย์ทุกบิต ✓ แปลว่าฝั่งส่งของเราคำนวณถูกต้องแล้วครับ

ชั้นที่ 3 — ไล่ทีละขั้น (ค) ฝั่งรับ กรณีจริงตามโจทย์: 110011100

คราวนี้เอาชุดที่โจทย์บอกว่ารับมาจริง คือ **110011100** มาหารด้วย **1101** ตัวเดิม **ไม่เติมศูนย์เพิ่ม** ทั้งหมด **6 รอบ** ครับ:

รอบ 1 · บิตนำ = 1 ⇒ XOR ด้วย G

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

รอบ 2 · บิตนำ = 0 ⇒ ข้าม

```
0 0 0 1 1 1 1 0 0
```

ไม่เปลี่ยน

รอบ 3 · บิตนำ = 0 ⇒ ข้าม

```
0 0 0 1 1 1 1 0 0
```

ไม่เปลี่ยน

รอบ 4 · บิตนำ = 1 ⇒ XOR ด้วย G

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

รอบ 5 · บิตนำ = 0 ⇒ ข้าม

```
0 0 0 0 0 1 0 0 0
```

ไม่เปลี่ยน

รอบ 6 · บิตนำ = 1 ⇒ XOR ด้วย G

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

ชั้นที่ 4 — คำตอบสุดท้าย

คำตอบข้อ 9

(ก) CRC ที่ต้องส่ง = 100 และ**ชุดข้อมูลที่ส่งออกไปจริง = 110010100**

(ข) ผลลัพธ์หลังหารที่ฝั่งรับคือ 000000101 ⟹ เศษ = 101 ซึ่งไม่ใช่ศูนย์

⟹ ข้อมูล 110011100 ที่รับมา “มี error” ครับ ต้องขอให้ต้นทางส่งใหม่

“The transmitted CRC is 100, so the frame sent is 110010100. Dividing the received word

110011100 by G = 1101 leaves a remainder of 101, which is not zero, so the received data contains an error and a retransmission is requested.”

ชั้นที่ 5 — ตรวจคำตอบด้วยวิธีที่สอง

ตรวจ 3 ทาง

ทางที่ 1 — หารซ้ำชุดที่ส่ง เอา 110010100 หารด้วย 1101 ได้เศษ **000** เป็นศูนย์ทุกบิต ✓ (ทำไว้ให้ดูแล้วในหัวข้อ (ข) ข้างบน) นี่ยืนยันว่า CRC = 100 ถูกต้อง

ทางที่ 2 — เทียบบิตต่อบิต ส่งไป 110010100 แต่รับได้ 110011100 ⟹ ต่างกันที่**ตำแหน่งที่ 6** (นับจากซ้าย) ซึ่งเป็นบิตพลิก 1 ตัว สอดคล้องกับที่ CRC จับได้ว่าเศษไม่เป็นศูนย์ ✓

ทางที่ 3 — ตรวจด้วยเครื่องมือ รันด้วย `tools/dig.py` : `crc("110010", "1101")` ได้ **("100", "110010100")** และ `crc_check("110011100", "1101")` ได้ **("101", False)** ✓ ตรงกับที่คำนวณมือทุกตัว

ชั้นที่ 6 — กับดักของข้อนี้

กับดัก · 4 อย่างที่ทำให้เสียคะแนนข้อนี้

1. ฝั่งรับเผลอเติมศูนย์อีก 3 ตัว — ถ้าทำแบบนั้นจะได้หาร 110011100 ต่อด้วย 000 ซึ่งเป็นคนละโจทย์เลยครับ ชุดที่รับมามี CRC ติดมาแล้ว พร้อมหารได้ทันที

2. เขียนเศษไม่ครบ 3 บิต — ถ้าหารแล้วได้ผลสุดท้ายลงท้ายด้วย 100 แล้วเผลอตัด 0 นำหน้าทิ้ง คำตอบจะสั้นผิดความยาว ต้องเขียน **3 บิตเสมอ**

3. ทำ XOR เป็นการลบแบบมีตัวยืม — ถ้าเจอ 0 กับ 1 แล้วไปยืมจากหลักซ้าย นั่นคือวิธีลบของ EP02 ไม่ใช่ modulo-2 ครับ ผลจะเพี้ยนตั้งแต่รอบแรก

4. ตอบว่า “ผิดที่บิตที่ 6” — ห้ามตอบแบบนี้เด็ดขาดครับ CRC **ตรวจได้ว่ามี error แต่บอกตำแหน่งไม่ได้** ที่พี่รู้ว่าเป็นบิตที่ 6 เพราะพี่บังเอิญมีชุดที่ส่งไว้เทียบ ซึ่งฝั่งรับจริง ๆ ไม่มี ถ้าโจทย์ถามตำแหน่ง นั่นคือโจทย์ Hamming ไม่ใช่ CRC

หัวใจของข้อ 9: ฝั่งส่ง “เติมศูนย์ r ตัวแล้วเอาเศษไปแทน” · ฝั่งรับ “หารทั้งก้อนแล้วดูว่าเศษเป็นศูนย์ไหม” เท่านั้นเองครับ

11 เฉลย Assignment ข้อ 10 — Hamming code (ข้อที่ยากที่สุด)

ข้อนี้คือข้อที่ยาวที่สุดของ Assignment ครับ พี่จะทำครบทั้ง**ฝั่งส่ง (เข้ารหัส)** และ **ฝั่งรับ (ตรวจและหาตำแหน่ง)** พร้อมตารางไล่ทุกกลุ่มพาริตี ไม่ข้ามขั้นไหนเลย

ชั้นที่ 1 — โจทย์ต้นฉบับและสิ่งที่โจทย์ซ่อนไว้

Assignment ข้อ 10 EES271 Assignment #2 · ข้อ 10

> Encode 7-bit data 1011001 in Hamming code using even parity. Also verify that the received data 10101101110 at the receiver has an error or not.

แปล: จงเข้ารหัส Hamming ให้ข้อมูล 7 บิต 1011001 ด้วย even parity แล้วตรวจว่าข้อมูล 10101101110 ที่ฝั่งรับได้มามี error หรือไม่ครับ

โจทย์ซ่อนอะไรไว้

1. คำว่า “7-bit data” เป็นตัวกำหนดทุกอย่าง — $d = 7$ บังคับให้ $p = 4$ และรหัสยาว 11 บิต ถ้าอ่านข้ามแล้วใช้โครง 7 บิต (ซึ่งเป็นของ $d = 4$) จะผิดตั้งแต่ต้น

2. “even parity” ต้องอ่านให้เห็น — ถ้าเป็น odd คำตอบของ parity ทุกตัวจะพลิกหมด

3. ข้อมูลที่รับมายาว 11 บิต เท่ากับความยาวรหัสพอดี $\Longrightarrow$ ยืนยันว่าเป็นรหัส Hamming ของ $d = 7$ จริง $\Longrightarrow$ ต้องตรวจ **4 กลุ่ม** คือ C1 C2 C4 C8

4. สองท่อนนี้เป็นคนละโจทย์กัน — ท่อนแรกให้เข้ารหัส ท่อนที่สองให้ตรวจของที่รับมา ฝั่งรับ**ไม่รู้**ว่าต้นทางส่งอะไรมา มันตัดสินจากกฎพาริตีล้วน ๆ ครับ

ชั้นที่ 2 — เตรียมก่อนลงมือ

หาจำนวน parity ด้วยสูตร $2^p \geq d + p + 1$ ก่อนเสมอ เลือกวิธีนี้เพราะถ้าจำนวน parity ผิด ตำแหน่งจะเลื่อนหมดทั้งรหัสครับ:

หาจำนวนบิตพาริตีของ d = 7

```
ลอง p = 3 :   2^3 = 8    แต่ d+p+1 = 7+3+1 = 11
              8 < 11   ->   ไม่พอ
ลอง p = 4 :   2^4 = 16   และ d+p+1 = 7+4+1 = 12
              16 >= 12   ->   พอ
-------------------------------------------------
ใช้ p = 4   ->   ความยาวรหัส n = d + p = 7 + 4 = 11 บิต
```

วางตำแหน่ง — parity อยู่ที่ตำแหน่งกำลังของ 2 คือ **1, 2, 4, 8** · ที่เหลือคือ **3, 5, 6, 7, 9, 10, 11** เก็บข้อมูล 1011001 เรียงตามลำดับเดิมครับ



วางข้อมูล 1011001 ลงในช่องฟ้าตามลำดับ · ช่องส้ม (P) ยังว่างอยู่

บิตข้อมูลตัวที่	ตัวที่ 1	ตัวที่ 2	ตัวที่ 3	ตัวที่ 4	ตัวที่ 5	ตัวที่ 6	ตัวที่ 7
ไปอยู่ตำแหน่งที่	3	5	6	7	9	10	11
ค่าของบิต	1	0	1	1	0	0	1

ชั้นที่ 3 — ไล่ทีละขั้น (ก) ฝั่งส่ง: คำนวณ parity ทั้ง 4 ตัว

ตอนเข้ารหัส แต่ละ parity นับเฉพาะ**บิตข้อมูล**ที่มันคุมครับ (ยังไม่นับ parity ตัวอื่น เพราะยังไม่รู้ค่า) แล้วเติมค่าที่ทำให้จำนวน 1 ในกลุ่มนั้นเป็น**คู่** ตามที่โจทย์สั่งว่า even parity:

บิตพาริตี	คุมตำแหน่งที่เป็นข้อมูล	บิตในตำแหน่งนั้น	จำนวน 1	ต้องเติมอะไร
P1	3, 5, 7, 9, 11	1 0 1 0 1	3	**1**
P2	3, 6, 7, 10, 11	1 1 1 0 1	4	**0**
P4	5, 6, 7	0 1 1	2	**0**
P8	9, 10, 11	0 0 1	1	**1**

ทำไม P8 ถึงคุมแค่ 4 ตำแหน่ง

เพราะรหัสยาวแค่ 11 บิตครับ — P8 คุมตำแหน่งที่มีบิตค่า 8 ติด 1 คือ 8, 9, 10, 11, 12, 13, 14, 15 แต่รหัสจบที่ 11 เพราะฉะนั้นเหลือแค่ **8, 9, 10, 11** ไม่ได้แปลว่าทำตกนะครับ



เติม parity ครบทั้ง 4 ตัวแล้ว ได้รหัส Hamming สมบูรณ์ 10100111001

ชั้นที่ 3 — ไล่ทีละขั้น (ข) ฝั่งรับ: ตรวจ 10101101110



รหัสที่รับมา 10101101110 วางลงตำแหน่ง 1 ถึง 11

จุดที่ต่างจากตอนเข้ารหัส — อ่านให้ดี

ตอนเข้ารหัส เรานับ**เฉพาะบิตข้อมูล** เพราะเรากำลังจะ**สร้าง** parity ขึ้นมา

แต่ตอนตรวจ เรานับ**ทุกตำแหน่งในกลุ่ม รวม parity bit ของกลุ่มนั้นเข้าไปด้วย** เพราะเรากำลัง**ตรวจสอบ**ว่ากฎ “จำนวน 1 เป็นคู่” ยังเป็นจริงอยู่ไหมครับ — ตรงนี้พลาดกันเยอะที่สุด

การตรวจ	ตำแหน่งที่ต้องนับ	บิตในตำแหน่งนั้น	จำนวน 1	ผลการตรวจ (even parity)
C1	1, 3, 5, 7, 9, 11	1 1 1 0 1 0	4	คู่ ⟹ ผ่าน (C1 = 0)
C2	2, 3, 6, 7, 10, 11	0 1 1 0 1 0	3	คี่ ⟹ ผิด (C2 = 1)
C4	4, 5, 6, 7	0 1 1 0	2	คู่ ⟹ ผ่าน (C4 = 0)
C8	8, 9, 10, 11	1 1 1 0	3	คี่ ⟹ ผิด (C8 = 1)

ชั้นที่ 3 — ไล่ทีละขั้น (ค) ประกอบ syndrome เป็นเลขที่อยู่

จากตารางข้างบน กลุ่มที่ **ไม่ผ่าน** คือ **C2, C8** ส่วนกลุ่มที่ผ่านคือ C1, C4 ครับ

เอา**เลขตำแหน่งของกลุ่มที่ไม่ผ่าน**มาบวกกันครับ — ทำได้เพราะเลขตำแหน่งของ parity ที่คุมบิตหนึ่ง ๆ ประกอบกันเป็นเลขฐานสองของตำแหน่งนั้นพอดี (ดูหัวข้อ 7 เรื่องความสวยงามของ Hamming):

ประกอบ syndrome

```
2 + 8 = 10
```

```
syndrome = 10  ->  ผิดที่ตำแหน่งที่ 10
```

อ่าน syndrome เป็นเลขฐานสองก็ได้ผลเดียวกัน

```
C1=0  C2=1  C4=0  C8=1
```

```
1010 ฐานสอง = 10 ฐานสิบ
```



syndrome = 10 ⟹ บิตที่ผิดคือช่องกรอบแดง ตำแหน่งที่ 10

ชั้นที่ 4 — คำตอบสุดท้าย

คำตอบข้อ 10

(ก) เข้ารหัส: ข้อมูล 1011001 ⟹ รหัส Hamming (even parity) = 10100111001 (11 บิต · P1 = 1, P2 = 0, P4 = 0, P8 = 1)

(ข) ตรวจ 10101101110: กลุ่มที่ไม่ผ่านคือ C2, C8 ⟹ syndrome = 2 + 8 = 10

⟹ ข้อมูลที่รับมา "มี error" และอยู่ที่ตำแหน่งที่ 10 (นับจากซ้าย เริ่มที่ 1)

แก้โดยพลิกบิตนั้น ตำแหน่งที่ 10 เดิมเป็น 1 ⟹ เปลี่ยนเป็น 0 ⟹ ได้ **10101101100** ซึ่งตรวจซ้ำแล้วได้ syndrome = 0 ✓

"Encoding 1011001 gives the Hamming code 10100111001. For the received word 10101101110, checks C2 and C8 fail, so the syndrome is 1010 = 10 and the received data has an error in bit position 10."

ชั้นที่ 5 — ตรวจคำตอบด้วยวิธีที่สอง

ตรวจ 3 ทาง

ทางที่ 1 — เอารหัสที่เข้าไว้มาตรวจแบบฝั่งรับ เอา 10100111001 มาไล่ C1 C2 C4 C8 ต้องได้จำนวน 1 เป็นคู่ทั้งสี่กลุ่ม ⟹ syndrome = **0** ✓ ยืนยันว่า parity ที่คำนวณไว้ถูกต้อง

ทางที่ 2 — แก้บิตแล้วตรวจซ้ำ พลิกตำแหน่งที่ 10 ของ 10101101110 ได้ 10101101100 เอามาตรวจใหม่ ได้ syndrome = **0** ✓ แปลว่าตำแหน่งที่หาได้สอดคล้องกับกฎทุกข้อ

ทางที่ 3 — ตรวจด้วยเครื่องมือ รันด้วย `tools/dig.py` : `hamming_encode("1011001")` ได้ **"10100111001"** และ `hamming_check("10101101110")` ได้ **(10, "10101101100")** ✓ ตรงกับที่คำนวณมือทุกตัว

ชั้นที่ 6 — กับดักของข้อนี้

กับดัก · 5 อย่างที่ทำให้เสียคะแนนข้อนี้

1. ใช้โครง 7 บิตแทน 11 บิต — ข้อมูล 7 บิตต้องใช้รหัส **11** บิต ที่เห็นเลข 7 ในสไลด์บ่อย ๆ นั่นคือกรณี $d = 4$ ครับ อย่าจำสลับ

2. ตอนตรวจลืมนับ parity bit ของกลุ่มตัวเอง — ผลจะกลับด้านทุกกลุ่ม แล้ว syndrome จะชี้ไปผิดที่

3. นับตำแหน่งเริ่มที่ 0 — ต้องเริ่มที่ **1** ครับ ถ้าเริ่มที่ 0 จะได้คำตอบว่าผิดที่ตำแหน่งที่ 9 ซึ่งผิด

4. เอา syndrome ไปนับจากขวา — ตำแหน่งที่ 10 นับ**จากซ้าย**เสมอครับ

5. คิดว่ารหัสที่รับมาต้องเป็นรหัสที่เราเพิ่งเข้าไว้ — ลองเทียบดูก็ได้ครับ 10100111001 กับ 10101101110 ต่างกันหลายตำแหน่ง แปลว่าโจทย์ท่อนที่สอง**เป็นคนละชุดข้อมูล** ฝั่งรับไม่มีทางรู้ว่าต้นทางส่งอะไร มันตัดสินจากกฎพาริตีอย่างเดียว ⟹ ตอบตามกฎว่า **ผิดที่ตำแหน่งที่ 10** ครับ

ข้อสังเกตเพิ่มเติมสำหรับคนอยากเข้าใจลึก

Hamming code รับประกันความถูกต้องของการ “ชี้ตำแหน่ง” **ภายใต้สมมติฐานว่าพลิกไม่เกิน 1 บิต** เท่านั้นครับ ถ้าพลิกจริง ๆ มากกว่านั้น syndrome ก็ยังคำนวณออกมาได้เลขหนึ่ง และจะชี้ไปที่ตำแหน่งที่อาจไม่ได้ผิดเลย

เพราะฉะนั้นคำตอบของข้อสอบคือ **“มี error · syndrome = 10 · ผิดที่ตำแหน่งที่ 10”** ซึ่งเป็นการตอบตามกระบวนการที่โจทย์ให้ใช้ ถูกต้องครับ

หัวใจของข้อ 10: ตอนเข้ารหัส “นับเฉพาะบิตข้อมูล” · ตอนตรวจ “นับทั้งกลุ่ม” · แล้วเอาเลขตำแหน่งของกลุ่มที่พังมาบวกกัน คือคำตอบ

12 กับดักรวมของทั้ง EP + สรุปหน้าเดียว

กับดัก 10 อันที่กินคะแนน EP นี้มากที่สุด

#	เรื่อง	อาการที่เจอบ่อย	วิธีกันพลาด
1	**BCD**	อ่าน BCD เป็นเลขฐานสองก้อนเดียว ($0100\,0111_{\text{BCD}}$ ตอบ 71 แทนที่จะเป็น 47)	**ดูตัวห้อยก่อนเสมอ** เจอ BCD ⟹ หั่นทีละ 4 บิตทันที
2	**BCD**	ปล่อยให้มีก้อน 1010–1111 โผล่ในคำตอบ	เขียนเสร็จ**ไล่ตรวจทุกก้อนว่าอยู่ในช่วง 0000–1001** ไหม
3	**BCD**	ข้ามหลักที่เป็นเลข 0 ไป (8013 เขียนเหลือ 3 ก้อน)	ทำนายความยาวไว้ก่อน: จำนวนหลัก × 4 = จำนวนบิต
4	**Gray**	สับสนสองทิศ — ทำขากลับด้วยสูตรขาไป	**ให้ $_2$ ขอ Gray = ขาไป (คู่จากแถวบนล้วน) · ให้ Gray ขอ $_2$ = ขากลับ (ทบต่อกัน)**
5	**Gray**	เอา Gray ไปคูณน้ำหนัก 8-4-2-1 ตรง ๆ	Gray เป็น **unweighted** ⟹ ต้องแปลงเป็นฐานสองก่อนถึงจะคิดค่าได้
6	**Parity**	สลับ even กับ odd	**วงกลมคำว่า even/odd ในโจทย์ก่อนลงมือทุกครั้ง**
7	**Parity**	ตอนตรวจลืมนับ parity bit รวมเข้าไปด้วย	ฝั่งรับนับ**ทั้งชุด**เสมอ ไม่แยก parity ออกก่อน
8	**CRC**	ฝั่งรับเผลอเติมศูนย์อีกรอบ · เติมศูนย์ผิดจำนวน	r = ความยาว G **ลบ 1** · ฝั่งรับ**ห้ามเติมศูนย์**
9	**CRC**	ทำ XOR เป็นการลบแบบมีตัวยืม	ท่องว่า $1 \oplus 1 = 0$ **ไม่ใช่ 10** — ไม่มียืมไม่มีตัวทด
10	**Hamming**	ตอนเข้ารหัสกับตอนตรวจใช้กติกาสลับกัน	**เข้ารหัส = นับเฉพาะบิตข้อมูล · ตรวจ = นับทั้งกลุ่ม**

ตารางเปรียบเทียบ 3 วิธีตรวจ error — ข้อสอบชอบถามตรงนี้

หัวข้อ	Parity	CRC	Hamming code
บิตส่วนเกิน	1 บิต	r บิต (r = ความยาว G − 1)	p บิต โดย $2^p \geq d + p + 1$
วิธีคำนวณ	นับจำนวนเลข 1	หารด้วย G แบบ modulo-2	parity หลายตัวคุมกลุ่มที่ซ้อนทับกัน

หัวข้อ	Parity	CRC	Hamming code
บอกได้ว่าผิด	**ได้**	**ได้**	**ได้**
บอกตำแหน่งได้	**ไม่ได้**	**ไม่ได้**	**ได้**
แก้ได้เอง	**ไม่ได้**	**ไม่ได้**	**ได้ (บิตเดียว)**
ข้อจำกัด	พลิก 2 บิตจับไม่ได้	จับได้เยอะแต่ไม่ครบทุกกรณี	พลิก 2 บิตจะชี้ผิดที่
คำอังกฤษที่ต้องใช้	detect only	detect only	detect **and correct**

สรุปหน้าเดียว — EP04 รหัสและการตรวจจับข้อผิดพลาด

BCD — เขียนเลขฐานสิบทีละหลัก

หนึ่งหลักฐานสิบ = 4 บิต ใช้ได้เฉพาะ 0000 ถึง 1001 · **1010–1111 คือ 6 รหัสต้องห้าม** เพราะหลักเดียวมีค่าเกิน 9 ไม่ได้

น้ำหนักในแต่ละก้อนคือ **8-4-2-1** · มีจุดทศนิยมให้**ยืนที่จุดแล้วหั่นออกไปสองทาง**

BCD ไม่เท่ากับเลขฐานสองปกติ ยกเว้นเลขหลักเดียวที่ไม่เกิน 9 · ตัวอย่าง $47_{10} = 0100\,0111_{\text{BCD}}$ แต่ $= 101111_2$

Gray code — เปลี่ยนทีละ 1 บิตเสมอ

unweighted — ห้ามคูณน้ำหนักตรง ๆ ต้องแปลงเป็นฐานสองก่อน

binary ⟹ Gray: MSB ลอกมา แล้ว **XOR ทุกคู่ที่ติดกันของต้นฉบับ** (คู่จากแถวบนล้วน)

Gray ⟹ binary: MSB ลอกมา แล้ว **เอาผลตัวที่เพิ่งได้ XOR กับ Gray ตัวถัดไป** (ทบต่อกันทีละก้าว)

ใช้กับ **shaft encoder** เพราะ binary ตอน $0111 \to 1000$ เปลี่ยน 4 บิตพร้อมกัน อ่านพลาดได้

ASCII — 7 บิต 128 ตัว

'0'–'9' = 30h–39h · **'A'–'Z' = 41h–5Ah** · **'a'–'z' = 61h–7Ah** · ตัวเล็ก = ตัวใหญ่ + 20h

00h–1Fh = control codes (พิมพ์ไม่ออก) · 20h–7Fh = printable · **ตัวอักษร '7' (37h) ไม่ใช่ค่าตัวเลข 7 (07h)**

Parity — บิตเดียว ตรวจได้ แก้ไม่ได้

even parity: นับ 1 ได้คู่ ⟹ เติม 0 · นับ 1 ได้คี่ ⟹ เติม 1 · **odd parity ตอบตรงข้ามกับ even เสมอ**

จับได้เฉพาะบิตพลิกจำนวนคี่ — พลิก 2 บิตแล้วจำนวน 1 กลับมาเท่าเดิม จับไม่ได้

ฝั่งรับนับ **ทั้งชุดรวม parity bit** · ตอบได้แค่ “มี/ไม่มี error” **ห้ามตอบตำแหน่ง**

CRC — 4 ขั้นฝั่งส่ง 2 ขั้นฝั่งรับ

ฝั่งส่ง: ① ตกลง G ② เติมศูนย์ r **= ความยาว G − 1** ตัว ③ หารแบบ modulo-2 ④ **เอาเศษไปแทนที่ศูนย์แล้วส่ง**

ฝั่งรับ: หารทั้งก้อนด้วย G ตัวเดิม (**ไม่เติมศูนย์เพิ่ม**) ⟹ เศษ = 0 ผ่าน · เศษ ≠ 0 มี error

modulo-2 = XOR ไม่มียืม ไม่มีตัวทด · ทุกรอบตัดสินจาก**บิตนำ**อย่างเดียว (1 ⟹ XOR · 0 ⟹ ข้าม) · เศษต้องยาว r บิตเสมอ

Hamming code — บอกตำแหน่งได้ด้วย

จำนวนพาริตี: $2^p \geq d + p + 1 \Longrightarrow d = 4$ ได้ $p = 3$ (รหัส 7 บิต) · $d = 7$ ได้ $p = 4$ (รหัส 11 บิต)

ตำแหน่งกำลังของ 2 (1, 2, 4, 8) เก็บ parity ที่เหลือเก็บ data · **นับตำแหน่งเริ่มที่ 1**

Pk คุมทุกตำแหน่งที่เลขตำแหน่งมีบิตค่า k ติด 1 (P1 ⟹ 1,3,5,7,9,11 · P2 ⟹ 2,3,6,7,10,11 · P4 ⟹ 4,5,6,7 · P8 ⟹ 8,9,10,11)

เข้ารหัส: นับเฉพาะบิตข้อมูลในกลุ่ม แล้วเติมให้เป็นคู่ · **ตรวจ:** นับ**ทั้งกลุ่ม** รวม parity bit

syndrome = ผลบวกเลขตำแหน่งของกลุ่มที่ไม่ผ่าน = ตำแหน่งของบิตที่พัง · syndrome = 0 ⟹ ไม่มี error · **แก้ได้จริงแค่บิตเดียว**

เช็คลิสต์ก่อนสอบ — ตอบได้ทุกข้อถือว่าพร้อม

ติ๊ก	คำถามที่ต้องตอบได้ทันทีโดยไม่ต้องเปิดชีท
☐	บอกได้ไหมว่า **6 รหัสต้องห้ามของ BCD** คืออะไรบ้าง และ**ทำไม**ถึงใช้ไม่ได้
☐	เขียน 47_{10} ทั้งแบบ BCD และแบบฐานสองปกติ แล้วอธิบายว่าทำไมไม่เหมือนกัน
☐	บอกได้ไหมว่า binary $0111 \rightarrow 1000$ เปลี่ยนกี่บิต และ Gray เปลี่ยนกี่บิต
☐	แปลง binary ⟹ Gray และ Gray ⟹ binary ได้ทั้งสองทาง โดย**ไม่สับสนทิศ**
☐	อธิบายได้ไหมว่าทำไม parity **จับบิตพลิก 2 ตัวไม่ได้** (ต้องยกตัวอย่างได้ด้วย)
☐	ท่อง 4 ขั้นฝั่งส่งของ CRC ได้ครบ และรู้ว่า r มาจากไหน
☐	หาร modulo-2 ได้โดย**ไม่เผลอยืมเลข** และรู้ว่าฝั่งรับห้ามเติมศูนย์
☐	หา p จาก $2^p \geq d + p + 1$ ได้ และรู้ว่า $d = 7$ ให้รหัส 11 บิต
☐	เขียนตารางว่า P1 P2 P4 P8 คุมตำแหน่งไหนบ้างได้เอง **โดยไม่ต้องท่อง** (ใช้เลขตำแหน่งฐานสอง)
☐	อธิบายได้ไหมว่า**ทำไมผลบวกเลขตำแหน่งของพาริตีที่ผิด ถึงชี้ตำแหน่งบิตเสียได้พอดี**
☐	ทำ Assignment ข้อ 9 และ 10 ใหม่ได้เอง**โดยไม่เปิดเฉลย**

ปิดท้าย EP04 · เกริ่น EP05

สี่ EP ที่ผ่านมาเราอยู่กับ **“ตัวเลข”** มาตลอดครับ — EP01 เขียนเลขในฐานต่าง ๆ · EP02 บวกลบคูณหารและเลขติดลบ · EP03 เลขทศนิยมแบบ floating point · และ EP04 นี้คือการเขียนของให้เป็นรหัส แล้วกันไม่ให้รหัสพังระหว่างทาง

ตอนนี้น้องจัดการเลข 0 กับ 1 ได้หมดแล้ว ทีนี้ถึงเวลาถามว่า วงจรจริง ๆ มันเอา 0 กับ 1 มาคิดยังไงครับ — ที่ผ่านมาพี่พูดว่า “วงจรคำนวณให้” มาตลอด แต่ยังไม่เคยเปิดฝาดูข้างในเลยว่ามันมีอะไร EP05 เราจะเจอ **ลอจิกเกต 7 ตัว** คือ NOT AND OR NAND NOR XOR XNOR ซึ่งเป็นชิ้นส่วนเล็กที่สุดที่ประกอบกันเป็นทุกอย่างในเครื่องคิดเลข คอมพิวเตอร์ และมือถือครับ ของใกล้ตัวที่จะเปิดเรื่องคือ**สวิตช์ไฟกับหลอดไฟ** — สวิตช์ต่ออนุกรมคือ AND ต่อขนานคือ OR ง่ายแค่นั้นจริง ๆ ครับ

นี่คือหน้าตาของสิ่งที่รออยู่ใน EP05



เกต AND — ออก 1 ก็ต่อเมื่อเข้า 1 ทั้งสองขา
เหมือนสวิตช์ไฟสองตัวต่ออนุกรมกัน
ต้องเปิดทั้งคู่ไฟถึงจะติดครับ

ขาเข้าอยู่ที่ (X, Y+12) และ (X, Y+28) · ขาออกอยู่ที่ (X+56, Y+20) — EP05 จะได้ต่อสายเองครับ

EP05 เปิดฝาวงจรให้ดูข้างในเป็นครั้งแรก

แหล่งอ้างอิงของ EP นี้

แหล่ง	หน้า / ข้อ	ครอบเนื้อหาส่วนไหน
EES 271-02 Operation and codes-2026 Rev4	หน้า 14–17	Binary Codes for Decimal Numbers · BCD และตัวอย่าง
EES 271-02 Operation and codes-2026 Rev4	หน้า 17–22	The Gray Code · การแปลงไป-กลับ · shaft encoder
EES 271-02 Operation and codes-2026 Rev4	หน้า 23–24	Character Codes · ตาราง ASCII อ้างอิง
EES 271-02 Operation and codes-2026 Rev4	หน้า 25–27	Parity Method for Error Detection
EES 271-02 Operation and codes-2026 Rev4	หน้า 28–31	Cyclic Redundancy Check · modulo-2 · ตัวอย่างฝั่งส่งและฝั่งรับ
EES 271-02 Operation and codes-2026 Rev4	หน้า 32–35	Hamming Code · สูตร · ตารางใครคุมใคร · ตัวอย่าง encode และ decode
EES271 Assignment #2 (2026)	ข้อ 4–10	เฉลยครบทุกข้อย่อยในหัวข้อ 8 ถึง 11

แหล่ง	หน้า / ข้อ	ครอบเนื้อหาส่วนไหน
Floyd, Digital Fundamentals 10th ed.	Chapter 2	Digital Codes · BCD · Gray code · ASCII · Error detection
เครื่องมือตรวจคำตอบ	`build/tools/dig.py`	ทุกคำตอบตัวเลขใน EP นี้ผ่านการรันตรวจด้วยฟังก์ชัน `to_bcd from_bcd to_gray from_gray parity_bit parity_ok crc crc_check hamming_encode hamming_check` จริง

จบ EP04 แล้วครับ — น้องเก็บทั้ง “วิธีเขียนของให้เป็นรหัส” และ “วิธีกันรหัสไม่ให้พัง” ครบแล้ว เจอกัน EP05 กับลอจิกเกตครับ