

# How to represent numbers for computer arithmetic?

Sign-magnitude, one's complement, and two's complement

# The obvious sign-magnitude idea creates avoidable problems

## Sign + magnitude

|   |   |   |   |    |
|---|---|---|---|----|
| 0 | 1 | 0 | 1 | +5 |
| 1 | 1 | 0 | 1 | -5 |

### What goes wrong?

- Addition must branch on four sign combinations
- Zero has two encodings: +0 and -0
- The ordinary unsigned adder is no longer enough

**Two's complement removes these special cases.**

# One's complement negates by flipping every bit

Positive value



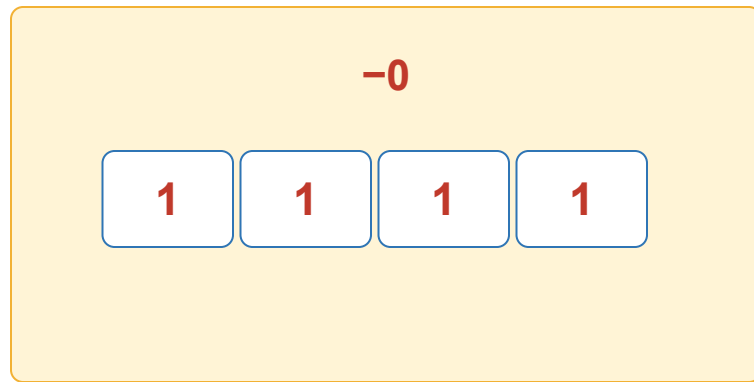
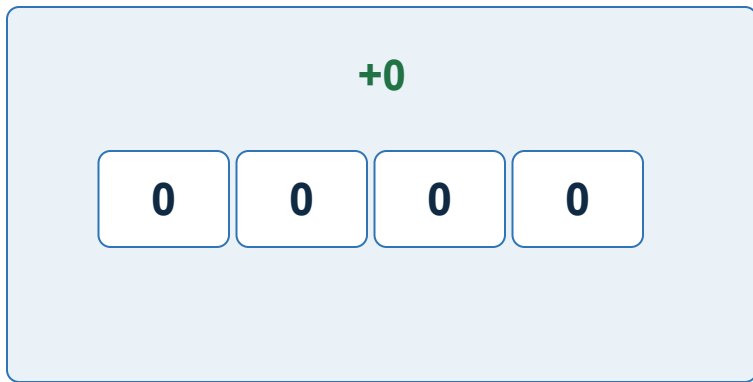
For N bits:  $-x$  is encoded as  $(2^N - 1) - x$

Range  $-(2^{N-1}-1)$  to  $+(2^{N-1}-1)$

A bit pattern and its inverse represent opposite values.

# One's complement has two encodings of zero

---



## Why does this happen?

Negating zero flips every 0 to 1. Both patterns mean zero, but their bits differ. Equality checks, (and normalization, hashing, and serialization) must therefore recognize or eliminate **-0**.

# One's-complement addition needs an end-around carry

Example:  $+5 + (-3) = +2$

$$\begin{array}{r} 0101 \\ + 1100 \\ \hline \text{carry } 1\ 0001 \end{array}$$

add the carry back

$$0001 + 1 = 0010$$

Why is this required?

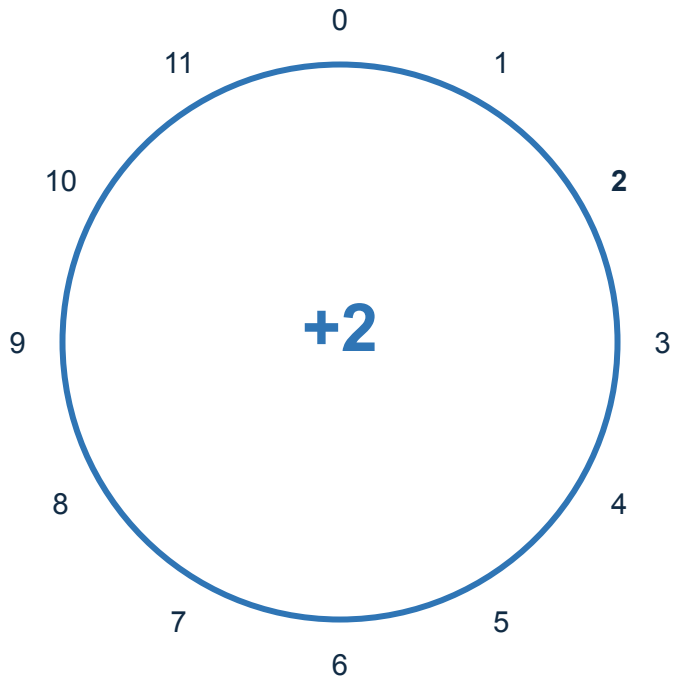
Arithmetic is effectively modulo  $2^N - 1$ .

Since  $2^N \equiv 1$ , a carry out of the top bit must be added into the low bit.

Ordinary fixed-width addition alone is not enough.

# Clock arithmetic makes wraparound meaningful

---



$$2 \equiv 14 \equiv -10 \pmod{12}$$

Different ordinary integers can name the same position on a finite ring.

**Forward 2 = forward 14 = backward 10**

# An N-bit machine already computes modulo $2^N$

## 3-bit example

$$4 + 5 = 9$$

$$100 + 101 =$$



Stored: 1

Why?

9 in binary is 1001



discard carry beyond 3 bits

The low 3 bits equal  $9 \bmod 8$ .

# The upper half of the ring becomes the negative range

---

| Bits | Unsigned | Signed |
|------|----------|--------|
| 000  | 0        | 0      |
| 001  | 1        | 1      |
| 010  | 2        | 2      |
| 011  | 3        | 3      |
| 100  | 4        | -4     |
| 101  | 5        | -3     |
| 110  | 6        | -2     |
| 111  | 7        | -1     |

If  $b \geq 2^{N-1}$ ,  $\text{signed}(b) = b - 2^N$

Example

$$101_2 = 5_u$$

$$5 - 8 = -3 \square$$

# One addition circuit supports both interpretations

---

$$110 + 111 \rightarrow 101$$

Unsigned interpretation

$$6 + 7 \rightarrow 5 \pmod{8}$$

Signed interpretation

$$(-2) + (-1) \rightarrow -3$$

Same bits in. Same bits out. Only the meaning changes.

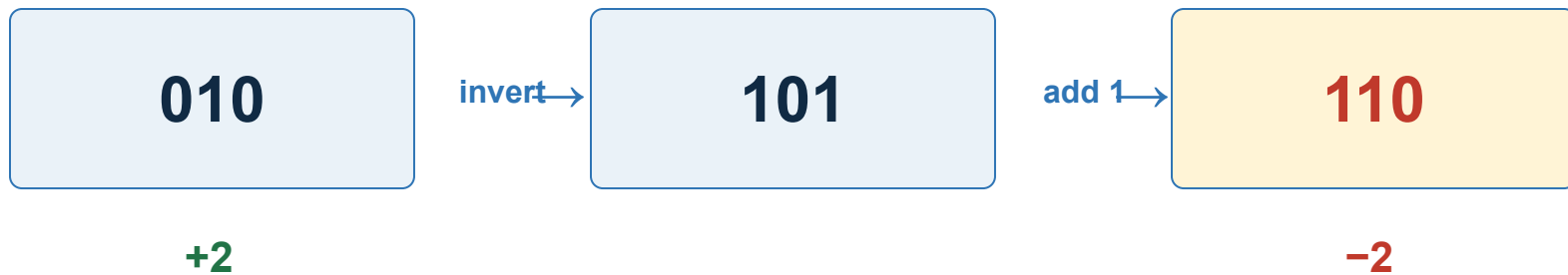
# Two's complement removes the arithmetic special cases

| Property       | Sign-magnitude       | One's complement  | Two's complement          |
|----------------|----------------------|-------------------|---------------------------|
| Negation       | flip sign bit        | invert all bits   | invert + 1                |
| Zero encodings | +0 and -0            | +0 and -0         | one zero                  |
| Addition       | sign-dependent logic | end-around carry  | ordinary N-bit adder      |
| N-bit range    | $\pm(2^{N-1}-1)$     | $\pm(2^{N-1}-1)$  | $-2^{N-1}$ to $2^{N-1}-1$ |
| Main issue     | four sign cases      | dual zero + carry | asymmetric range          |

The same modular adder works for signed and unsigned values.

## Negation is invert the bits, then add one

---



$$\sim x = 2^N - 1 - x, \text{ so } \sim x + 1 = 2^N - x \equiv -x \pmod{2^N}$$

## Read the top bit as a negative place value

---



-8

+4

+0

+1

$$-8 + 4 + 0 + 1 = -3$$

**General rule**

$$S = -b_{-1}2^{n-1}$$

$$+ \sum b_i 2^i$$

The most-significant bit has weight  $-2^{n-1}$ .

# Two's complement is the chosen representation

---

Fixed-width arithmetic → Ring modulo  $2^N$  → Signed interpretation

**The hardware performs modular arithmetic;  
the program decides how to interpret each bit pattern.**

## Quick check

What does  $11110110_2$  mean as an 8-bit signed integer?

**-10**

# Binary codes

Useful for special purposes, not general computer arithmetic

# Binary Coded Decimal Digits (BCD)

Used in digital calculators, financial databases etc.

# Binary codes change the symbols, not the information

---

Set to encode

0 1 2 3 4  
5 6 7 8 9

10 distinct decimal symbols



How many bits?

$$2^3 = 8$$

not enough

$$2^4 = 16$$

enough

Four bits provide 16 combinations: 10 are assigned and 6 remain unused.

# 8421 BCD encodes each decimal digit separately

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

## Example

**396**<sub>10</sub>



0011

3

1001

9

0110

6

**k decimal digits require 4k bits in BCD**

**1010 through 1111 are invalid BCD digits**

**Each 4-bit group must be decoded as one decimal digit.**

# A BCD word is not the binary value of the number

---

Decimal value

**185<sub>10</sub>**

**BCD representation**

**0001 1000 0101**

12 bits: one group per digit

**Pure binary representation**

**10111001**

8 bits: positional value in base 2

**Same number, different encodings—and different arithmetic rules.**

# BCD addition corrects every invalid digit sum by adding 0110

Add each pair as binary  $\rightarrow$  sum  $\leq 1001$  and no carry? **keep it** **otherwise + 0110**

**No correction**

$$0100 + 0101 = 1001$$

$$4 + 5 = 9$$

**Correction required**

$$0100 + 1000 = 1100$$

$$1100 + 0110 = 1\ 0010$$

$$4 + 8 = 12$$

Why 6? A binary carry is worth 16, but a decimal carry is worth 10:  $16 - 10 = 6$ .

# Multi-digit BCD addition corrects each decimal column

$$184 + 576 = 760$$

|            | hundreds | tens   | ones |
|------------|----------|--------|------|
| 184        | 0001     | 1000   | 0100 |
| +576       | 0101     | 0111   | 0110 |
| binary sum | 0111     | 1 0000 | 1010 |
| add 6      | —        | 0110   | 0110 |
| BCD result | 0111     | 0110   | 0000 |

ones:  $4 + 6 = 10 \rightarrow 0000$ , carry 1

tens:  $8 + 7 + 1 = 16 \rightarrow 0110$ , carry 1

# Many four-bit codes can represent the same ten digits

| Digit | BCD 8421 | 2421 | Excess-3 | 8, 4, -2, -1 |
|-------|----------|------|----------|--------------|
| 0     | 0000     | 0000 | 0011     | 0000         |
| 1     | 0001     | 0001 | 0100     | 0111         |
| 2     | 0010     | 0010 | 0101     | 0110         |
| 3     | 0011     | 0011 | 0110     | 0101         |
| 4     | 0100     | 0100 | 0111     | 0100         |
| 5     | 0101     | 1011 | 1000     | 1011         |
| 6     | 0110     | 1100 | 1001     | 1010         |
| 7     | 0111     | 1101 | 1010     | 1001         |
| 8     | 1000     | 1110 | 1011     | 1000         |
| 9     | 1001     | 1111 | 1100     | 1111         |

**Weighted codes: decode by summing the weights of the 1 bits.**

**Every code still leaves 6 of the 16 combinations unused.**

# ASCII code

Used to store everything other than numbers

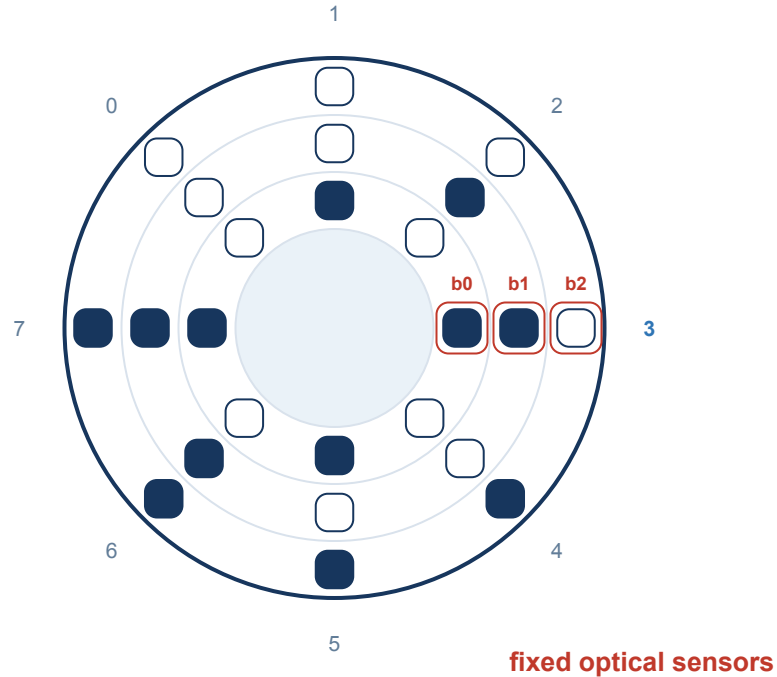
**Table 1.7***American Standard Code for Information Interchange (ASCII)*

| $b_4b_3b_2b_1$ | $b_7b_6b_5$ |     |     |     |     |     |     |     |
|----------------|-------------|-----|-----|-----|-----|-----|-----|-----|
|                | 000         | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
| 0000           | NUL         | DLE | SP  | 0   | @   | P   | '   | p   |
| 0001           | SOH         | DC1 | !   | 1   | A   | Q   | a   | q   |
| 0010           | STX         | DC2 | "   | 2   | B   | R   | b   | r   |
| 0011           | ETX         | DC3 | #   | 3   | C   | S   | c   | s   |
| 0100           | EOT         | DC4 | \$  | 4   | D   | T   | d   | t   |
| 0101           | ENQ         | NAK | %   | 5   | E   | U   | e   | u   |
| 0110           | ACK         | SYN | &   | 6   | F   | V   | f   | v   |
| 0111           | BEL         | ETB | '   | 7   | G   | W   | g   | w   |
| 1000           | BS          | CAN | (   | 8   | H   | X   | h   | x   |
| 1001           | HT          | EM  | )   | 9   | I   | Y   | i   | y   |
| 1010           | LF          | SUB | *   | :   | J   | Z   | j   | z   |
| 1011           | VT          | ESC | +   | ;   | K   | [   | k   | {   |
| 1100           | FF          | FS  | ,   | <   | L   |     | l   |     |
| 1101           | CR          | GS  | -   | =   | M   | ]   | m   | }   |
| 1110           | SO          | RS  | .   | >   | N   | ^   | n   | ~   |
| 1111           | SI          | US  | /   | ?   | O   | -   | o   | DEL |

# Gray Code

Used in physical sensors

# An absolute encoder reads shaft angle as a digital word



Opaque sector → 1

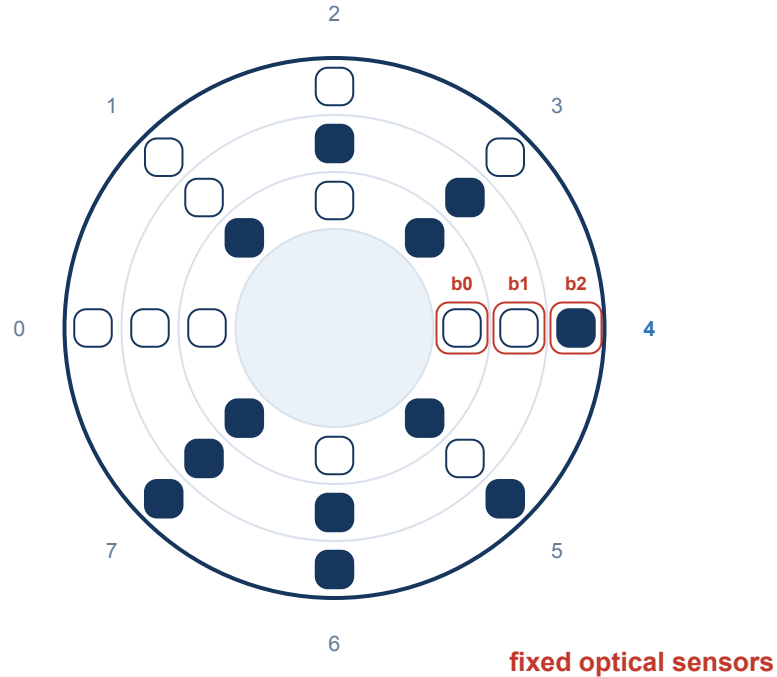
Transparent sector → 0

Sensor output

0 1 1 = position 3

Advance to animate →

# At the $3 \rightarrow 4$ boundary, every binary bit must change



**Before**  
 $011_2 = 3$   $\rightarrow$  **After**  
 $100_2 = 4$

**$b_2$ :  $0 \rightarrow 1$**

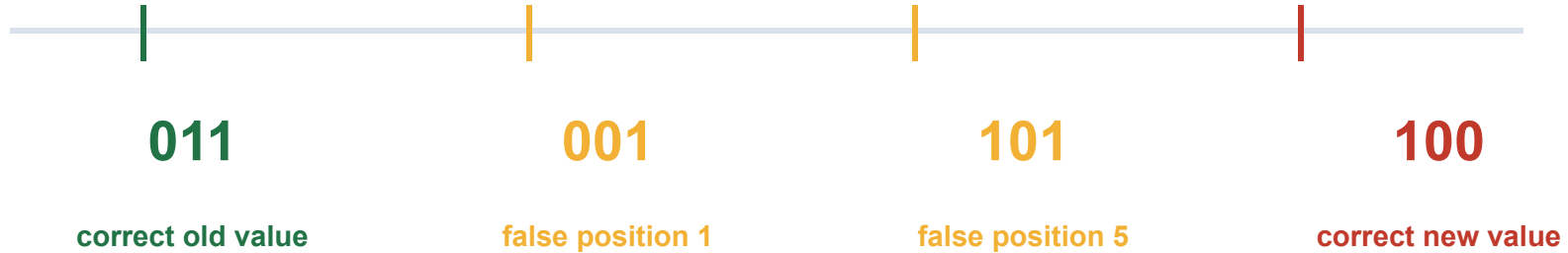
**$b_1$ :  $1 \rightarrow 0$**

**$b_0$ :  $1 \rightarrow 0$**

**Three physical sensors cannot switch at exactly the same instant.**

# Tiny sensor delays expose intermediate bit patterns

shaft crosses boundary



A slight timing mismatch can report a position far from both 3 and 4.

The error is caused by code geometry—not by incorrect arithmetic.

# A microscopic angular offset can become a large numeric error

true motion

**3** → **4**

possible sampled sequence

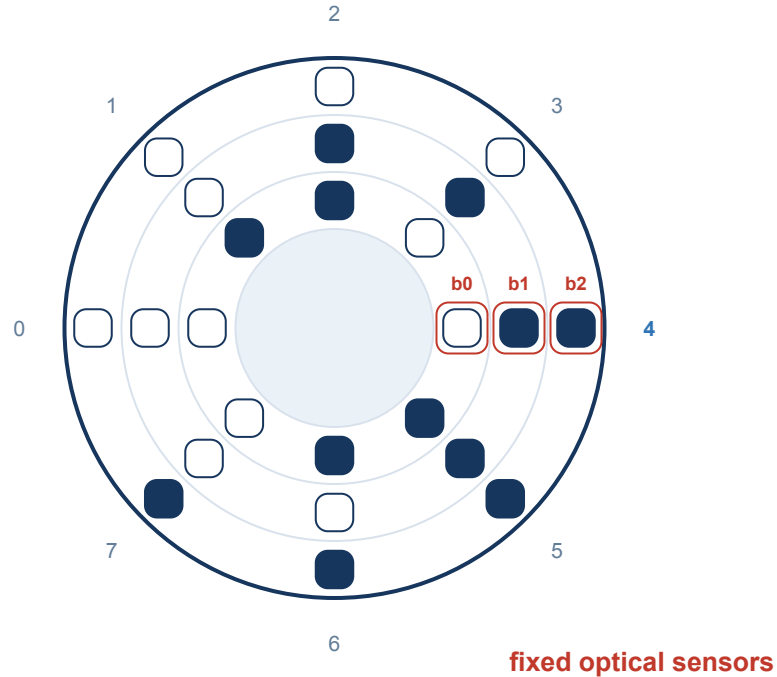
**3 → 1 → 5 → 4**

false backward and forward jumps

For an N-bit encoder, the worst transient error can span a large fraction of the full rotation.

# Gray code changes only one track at each boundary

---



Gray position 3

**010**

↓ only **b<sub>2</sub>** changes

**110**

Gray position 4

During the transition, the reading can only be the old or new adjacent code.

# Gray code trades arithmetic convenience for safer sensing

---

## Natural binary

**011 → 100**

**3 bits change**

- ✓ directly represents the number
- ✗ transition may be ambiguous

## Gray code

**010 → 110**

**1 bit changes**

- ✓ adjacent positions are robust
- ✗ convert to binary for arithmetic

**Encoding should match the physical failure mode.**