

You're invited to a special event

Considering a career in VLSI, chip design, circuit design, or computer architecture?

Join Apple and UC Berkeley EECS faculty for a special presentation about career paths in these exciting fields!

Afterwards, network with Apple engineers and UC Berkeley over burritos and boba.

New Silicon Initiative Fall Kickoff

Friday, September 5, 2025
11:00 a.m. – 1:00 p.m. PT
Banatao Auditorium

Check-in begins at 10:45 a.m. PT

Register by scanning or clicking the QR code. →





Announcements

- New Zoom link: cs61c.org/fa25/lecture-zoom
- Waiting on dept word about class expansion (+CEs); I'm hopeful
- Planning to enroll but need Ed access?
forms.gle/Qv2riacbtKCp2cZy6
- Discussion section format
 - On the fence about **Regular** vs. **Bridge** vs. **Video** discussion? Choose Regular/Bridge to be assigned a time that works for you.
 - Fill out welcome survey by **Friday (today) 11:59pm** to get assigned a regular time (if applicable). **All other students default to video.**
 - Can switch section times/formats until Add/Drop deadline (Week 4)
- 61C Scholars Pilot Program (self-identify on welcome survey)
 - Scholars-specific regular discussion + other activities/socials



CS61C

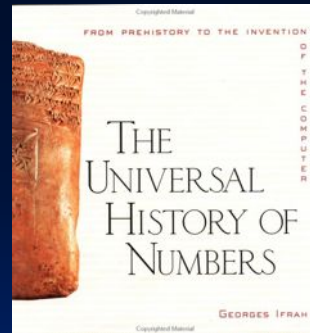
Great Ideas
in
Computer Architecture
(a.k.a. Machine Structures)



Teaching Professor
Dan Garcia

Number Representation

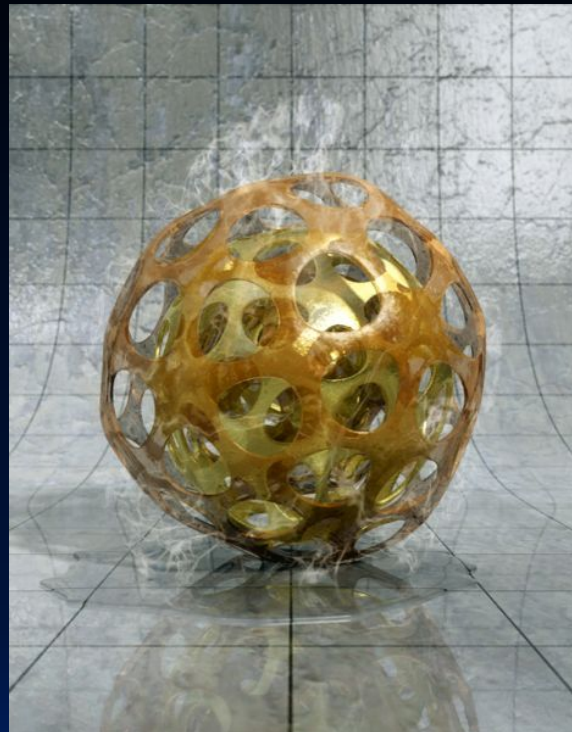
Great book ⇒
The Universal History of Numbers
by Georges Ifrah



Garcia, FA25



Digital data not necessarily born Analog...

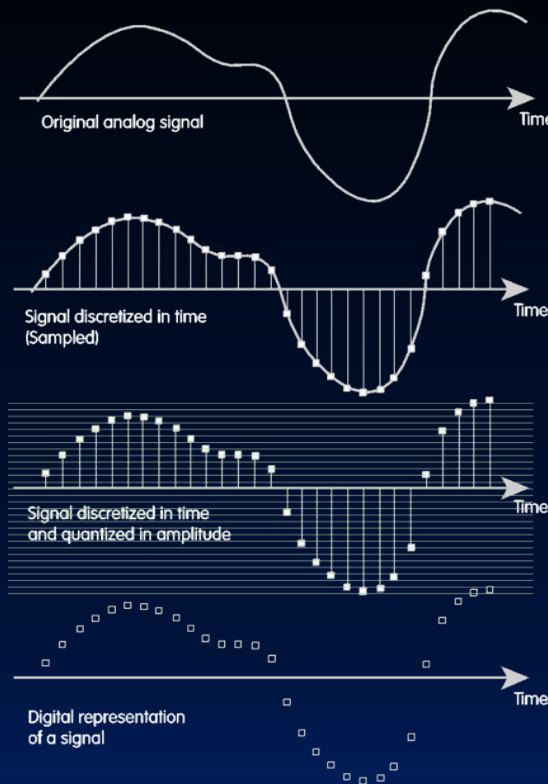


Data input: Analog \rightarrow Digital

Real world is analog!

To import analog information, we must do two things

- **Sample**
 - E.g., for a CD, every 44,100ths of a second, we ask a music signal how loud it is.
- **Quantize**
 - For every one of these samples, we figure out where, on a 16-bit (65,536 tic-mark) "yardstick", it lies.





Agenda

Binary, Decimal, Hex

- Binary, Decimal, Hex
- Integer Representations
- Sign-Magnitude, Ones' Complement
- Two's Complement
- Bias Encoding



Number vs Numeral

Numeral

A symbol or name that stands for a number
e.g., 4 , *four* , *quattro* , IV , IIII , ...

...and **Digits** are symbols that make numerals

Above the abstraction line

Abstraction Line

Below the abstraction line

Number

The "idea" in our minds...there is only ONE of these
e.g., *the concept of "4"*



Decimal: Base 10 (Ten) #s

Digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Example:

$$\mathbf{3271} = \mathbf{3271}_{10} = (\mathbf{3} \times 10^3) + (\mathbf{2} \times 10^2) + (\mathbf{7} \times 10^1) + (\mathbf{1} \times 10^0)$$



Base 2 (Two) #s, Binary

Digits: 0, 1 (binary digits → bits)

Example: Binary number "1101"

Convert to decimal:

$$\begin{aligned} \text{0b1101} &= 1101_2 = (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) \\ &= 8 + 4 + 0 + 1 \\ &= 13 \end{aligned}$$

Common binary shorthand:
0b1101



Base 16 (Sixteen) #s, Hexadecimal

Digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
10, 11, 12, 13, 14, 15

Example: Hexadecimal number "**A5**"

Convert to decimal:

$$\begin{aligned}\text{0xA5} &= \text{A5}_{16} = (10 \times 16^1) + (5 \times 16^0) \\ &= 160 + 5 \\ &= 165\end{aligned}$$

"Hex" for short.
Common hex
shorthand: 0xA5



Convert from Decimal to Binary

E.g., 13 to binary?

Start with the columns

~~13~~
~~5~~
~~1~~
0

$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
1	1	0	1

0b1101

Left to right, is (column) \leq number **n**?

- If yes, put how many of that column fit in **n**, subtract col * that many from **n**, keep going.
- If not, put 0 and keep going. (and Stop at 0)

Convert from Decimal to Hexadecimal

E.g., 165 to hexadecimal?
Start with the columns

165
5
0

$16^3 = 4096$	$16^2 = 256$	$16^1 = 16$	$16^0 = 1$
0	0	10(A)	5

0x00A5

0xA5

Left to right, is (column) \leq number **n**?

- If yes, put how many of that column fit in **n**, subtract col * that many from **n**, keep going.
- If not, put 0 and keep going. (and Stop at 0)

Nibbles and Bytes

Memorize this table.

- 4 Bits
 - 1 "Nibble"
 - 1 Hex Digit = 16 things
- 8 Bits
 - 1 "**Byte**"
 - 2 Hex Digits = 256 things

Dec	Hex	Bin
00	0	0000
01	1	0001
02	2	0010
03	3	0011
04	4	0100
05	5	0101
06	6	0110
07	7	0111
08	8	1000
09	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111



Convert Binary → Hexadecimal

Memorize this table.

- Binary → Hex? Easy!

- **Left-pad** with 0s
- (Group into full 4-bit values)
- Look it up
- `0b11110`
→ `0b00011110`
(→ `0b0001 1110`)
→ **`0x1E`**

- Hex → Binary? Easy!

- **Just look it up**
- `0x1E`
→ `0b00011110`
→ **`0b11110`** (drop leading 0s)

Dec	Hex	Bin
00	0	0000
01	1	0001
02	2	0010
03	3	0011
04	4	0100
05	5	0101
06	6	0110
07	7	0111
08	8	1000
09	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Binary odometer



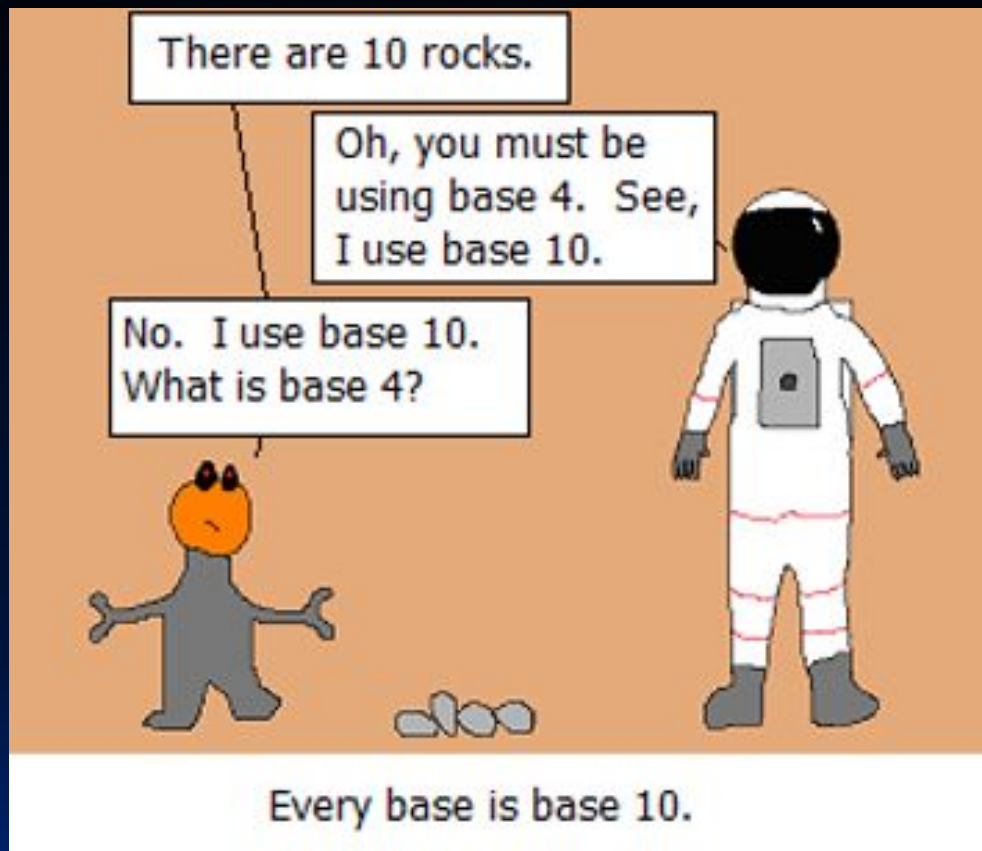
Car odometer



Which base do we use?

- **Decimal:** great for humans, especially when doing arithmetic
- **Hex:** if human looking at long strings of binary numbers, its much easier to convert to hex and see 4 bits/symbol
 - Terrible for arithmetic on paper
- **Binary:** what computers use
 - To a computer, numbers are always binary
 - Regardless of how number is written:
 - $32_{\text{ten}} == 32_{10} == 0x20 == 100000_2 == 0b100000$
- To avoid confusion:
 - Decimal: subscript "ten" or prefix (none)
 - Hex: subscript "hex" or prefix 0x
 - Binary: subscript "two" or prefix 0b

Every Base is Base 10...?



How many
rocks?



The computer knows it, too...

(more next time)

```
#include <stdio.h>
int main() {
    const int N = 1234;
    printf("Decimal: %d\n",N);
    printf("Hex:      %x\n",N);
    printf("Octal:     %o\n",N);

    printf("Literals (not supported by all compilers):\n");
    printf("0x4d2      = %d (hex)\n", 0x4d2);
    printf("0b10011010010 = %d (binary)\n", 0b10011010010);
    printf("02322        = %d (octal, prefix 0 - zero)\n", 0x4d2);
    return 0;
}
```

Output Decimal: 1234
 Hex: 4d2
 Octal: 2322
 Literals (not supported by all compilers):
 0x4d2 = 1234 (hex)
 0b10011010010 = 1234 (binary)
 02322 = 1234 (octal, prefix 0 - zero)

Garcia, FA25

BIG IDEA: Bits can represent anything!!

- Logical values? 1 bit
 - One possible convention: 0 → False, 1 → True
- Characters? Several options:
 - A, ..., Z: 26 letters → 5 bits ($26 \leq 32$)
 - ASCII: upper/lower case + punctuation → 7 bits → round to 1 byte
 - Unicode (www.unicode.com): standard code to cover all the world's languages ⇒ 8, 16, 32 bits
- Colors?
 - HTML color codes: 24 bits (3 bytes)
- Locations / addresses?
Commands?
 - IPv4 (32 bit), IPv6 (64 bit), etc.



California Gold

0xFDB515

Red (FD)

Green (B5)

Blue (15)

With N bits, you can represent at most 2^N things.



How many bits do you need to represent π (pi)?

- A. 1
- B. 9 ($\pi=3.14$, so 0.011". 001100)
- C. 64 (Macs are 64-bit machines)
- D. Every bit the machine has
- E. ∞

Number Representation: How many bits to represent π (Pi)?

25



Pop quiz??

1. How many "things" can be represented by 4 bits?
A. 4 **C.** 16
B. 8 **D.** 64
E. Something else
2. [no pollEV, just discuss]
How many bits do you need to represent π (pi)?
A. 1
B. 9 ($\pi=3.14$, so 0.011..." 001100)
C. 64 (Macs are 64-bit machines)
D. Every bit the machine has
E. ∞
3. [no pollEV, just discuss]
What does this particular 4-bit pattern represent?

1011

How many things can be represented using 4 bits?

0

1. How many "things" can be represented by 4 bits?
A. 4 **C.** 16
B. 8 **D.** 64
E. Something else
2. [no pollEV, just discuss]
How many bits do you need to represent π (pi)?
A. 1
B. 9 ($\pi=3.14$, so 0.011..." 001100)
C. 64 (Macs are 64-bit machines)
D. Every bit the machine has
E. ∞
3. [no pollEV, just discuss]
What does this particular 4-bit pattern represent?
1011

(A) 4

0

(B) 8

0

(C) 16

0

(D) 64

0

(E) Something else

0



Pop quiz??

1. How many "things" can be represented by 4 bits?
A. 4 **C.** 16 = 2^4
B. 8 **D.** 64
E. Something else
2. [no pollEV, just discuss]
How many bits do you need to represent π (pi)?
A. 1
B. 9 ($\pi=3.14$, so 0.011..." 001100)
C. 64 (Macs are 64-bit machines)
D. Every bit the machine has
E. ∞
3. [no pollEV, just discuss]
What does this particular 4-bit pattern represent?

1011



Integer Representations

- Binary, Decimal, Hex
- Integer Representations
- Sign-Magnitude, Ones' Complement
- Two's Complement
- Bias Encoding

How do we pick a representation for integers?

- Want a representation that supports common integer operations:
 - Add them
 - Subtract them
 - Multiply them
 - Divide them
 - Compare them ($<$, $=$, \neq , \leq , etc.)
- Example: $10 + 7 = 17$
 - 10, 7 can be represented with 4 bits:
 - Addition, subtraction just as you would in decimal!!
 - So simple to **add** in binary that we can build circuits to do it!
 - This design decision would make hardware simple!**
 - ...wait...

$$\begin{array}{r}
 11 \text{ carry bits} \\
 1010 \\
 + 0111 \\
 \hline
 10001
 \end{array}$$

What if "too big"? Overflow

- Strictly speaking, base 2 numerals have an ∞ number of digits.
 - With almost all being same (00...0 or 11...1) except rightmost digits
 - Just don't normally show leading digits
- ...000000001010
- However, **hardware has physical limits**. No infinite bits!
 - Common representations: 8 bits, 16 bits, 32 bits, 64 bits, ...
 - Again: With N bits, you can represent at most 2^N things.
 - If integer result of operation (+, -, *, /, >, <, =, etc.) cannot be represented by HW bits, we say **integer overflow** occurred

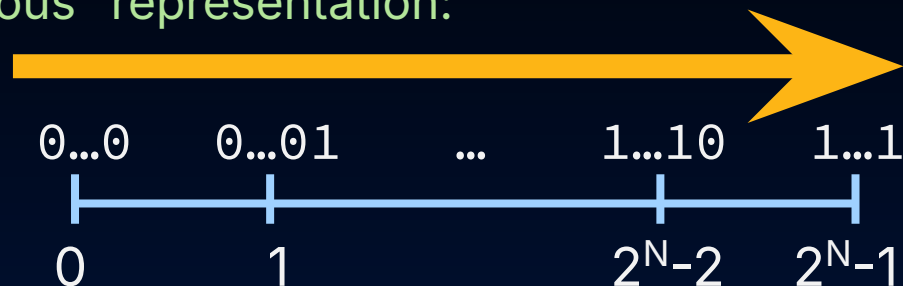


Integer overflow: The arithmetic result is outside the representable range.

Many Possible Number Representations

- So far, we have only discussed **unsigned numbers** (non-negative).
 - C's `uint8_t`, `uint16_t`, etc.: $[0, 2^N-1]$
 - Most computers use the "obvious" representation:

Binary odometer



- What about **signed numbers**? Need a way to represent **negative numbers**. Let's discuss a few:
 - Sign-Magnitude
 - Ones' Complement
 - Two's Complement (C23: the only signed integer rep permitted)
 - Bias Encoding (if time, otherwise review on your own)



Sign-Magnitude, Ones' Complement

- Binary, Decimal, Hex
- Integer Representations
- Sign-Magnitude, Ones' Complement
- Two's Complement
- Bias Encoding

Sign-Magnitude: Ain't No Free Lunch (tell story)

- Strawman ("obvious") solution:
 - Leftmost **sign bit**: 0 → +, 1 → -
 - Rest of bits: numerical value

Binary odometer



- Sign-magnitude is **rarely used**, due to many shortcomings:
 - Incrementing "binary odometer" increases then decreases values
 - Arithmetic circuit complicated: depends on signs same/different
 - **Two zeros** (how to compare??)
- Reasonable for signal processing,
not for general purpose computers

0x00000000 = +0_{ten}


0x80000000 = -0_{ten}

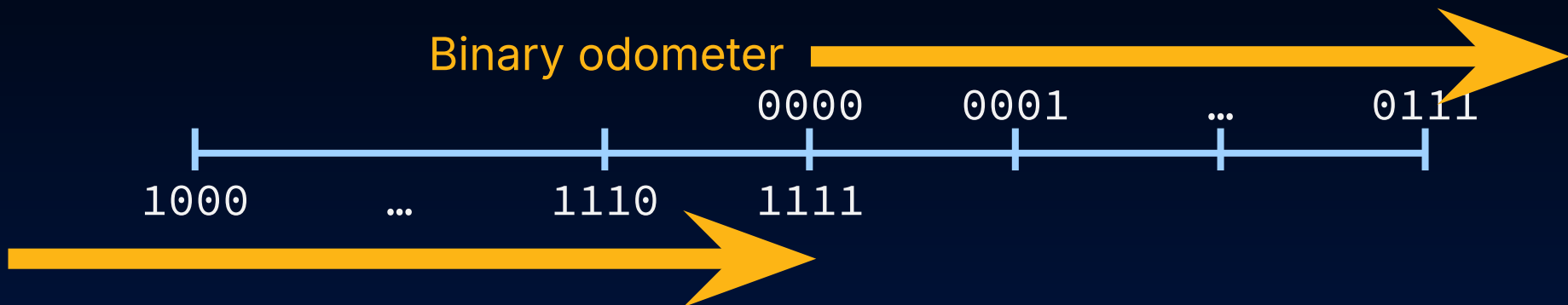
0b 1000 0000 ... 0000

Ones' Complement: Another try

- To represent a negative number, complement ("**flip**") the bits of its positive representation:

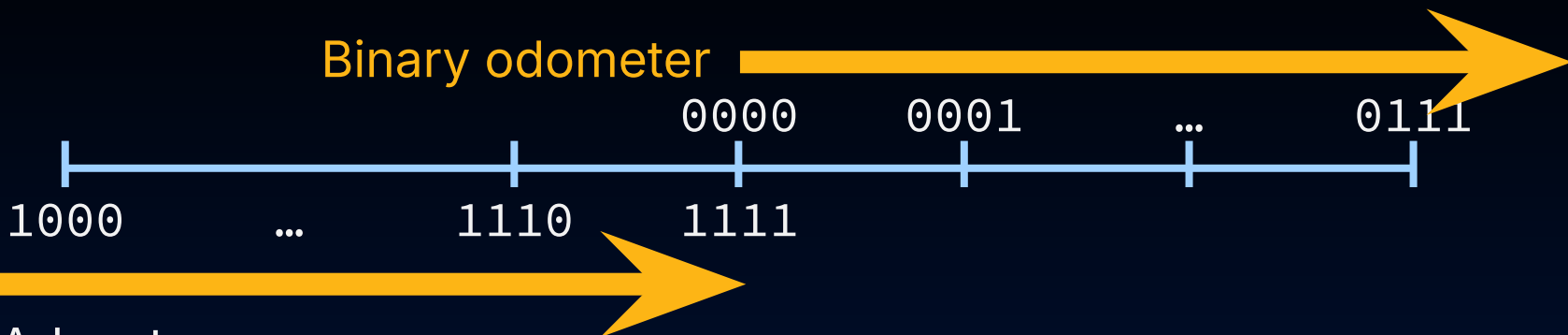
$$\begin{aligned}
 +7_{\text{ten}} &= 0b \ 0000 \ 0111 \\
 -7_{\text{ten}} &= 0b \ 1111 \ 1000
 \end{aligned}$$





- Observations:
 - Positive numbers: leading 0s
 - Negative numbers: leading 1s
- #s represented in N bits:
 - Zero: 2
 - Positive: $2^{N-1} - 1$
 - Negative: (same as positive)

Shortcomings of Ones' Complement?



- Advantages:
 - Leftmost bit ("**most significant bit**") is still effectively sign bit
 - Incrementing binary odometer consistent on the # line
- Some disadvantages still persist:
 - Still two zeros
 - Arithmetic still somewhat complicated (more later)
- While used for a while on some computer products
 - It's not currently used in current hardware



Two's Complement

- Binary, Decimal, Hex
- Integer Representations
- Sign-Magnitude, Ones' Complement
- Two's Complement
- Bias Encoding

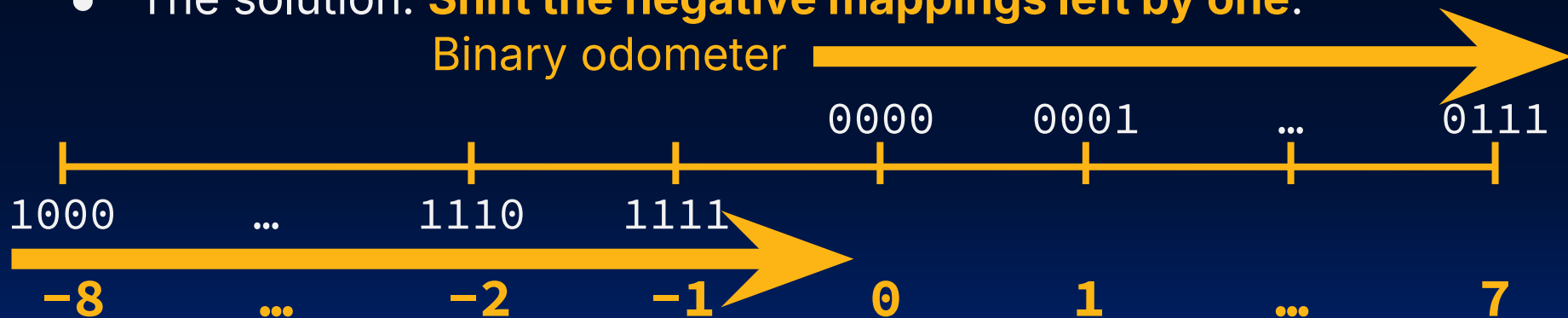
Two's Complement: C23 standard number rep.

- Ones' complement:



- The problem: Negative mappings "overlap" with the positive ones, creating the two 0s.
- The solution: **Shift the negative mappings left by one.**

Binary odometer





Arithmetic in Two's Complement is simple



- Advantages:

- Leftmost bit ("**most significant bit**") is still effectively sign bit
- Incrementing binary odometer consistent on the # line
- One zero, and one extra negative number (here, -8 vs 7)
- **Simple hardware for addition**

Decimal

$$\begin{array}{r} 5 \\ + -5 \\ \hline 0 \end{array}$$



Two's
complement

111 carry bits

$$\begin{array}{r} 0101 \\ + 1011 \\ \hline \text{dropped } 10000 \end{array}$$

Two's Complement: Formula

- Positive and negative numbers can be computed using the same formula:

0b1011

$$\begin{aligned}
 &= (1 \times -2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) \\
 &= -8 + 0 + 2 + 1 \\
 &= -5
 \end{aligned}$$

- Highest bit multiplied by neg power of 2!

0b0101

$$\begin{aligned}
 &= (0 \times -2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) \\
 &= 0 + 4 + 0 + 1 \\
 &= 5
 \end{aligned}$$

Two's Complement: Algorithm

At home: Prove algorithm is equivalent to formula!

- Positive and negative numbers can be computed using the same formula:

0b1011

$$\begin{aligned}
 &= (1 \times -2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) \\
 &= -8 + 0 + 2 + 1 \\
 &= -5
 \end{aligned}$$

- Highest bit multiplied by neg power of 2!

0b0101

$$\begin{aligned}
 &= (0 \times -2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) \\
 &= 0 + 4 + 0 + 1 \\
 &= 5
 \end{aligned}$$

- Hardware to convert positive to negative (& vice versa) is **simple**.

- Complement all bits
- Then add 1

$$5_{\text{ten}} \quad 0101 \rightarrow 1010 \rightarrow 1011 \quad -5_{\text{ten}}$$

$$-5_{\text{ten}} \quad 1011 \rightarrow 0100 \rightarrow 0101 \quad 5_{\text{ten}}$$



Two's Complement: C standard (as of 2025)

- Two's complement is the C23 standard number representation for signed integers.



- 2^{N-1} negatives
- 2^{N-1} non-negatives
 - 1 zero
 - How many positives?

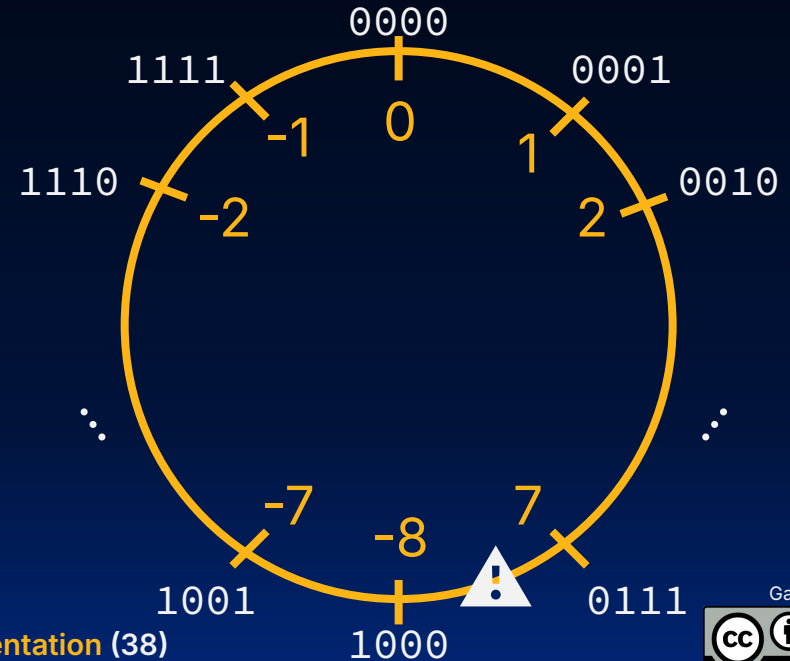
Two's Complement: Integer Overflow

- Two's complement is the C23 standard number representation for signed integers.



- 2^{N-1} negatives
- 2^{N-1} non-negatives
 - 1 zero
 - How many positives?

- Integer overflow** in two's complement can be conceptualized via a "number wheel":







Bias Encoding

- Binary, Decimal, Hex
- Integer Representations
- Sign-Magnitude, Ones' Complement
- Two's Complement
- Bias Encoding

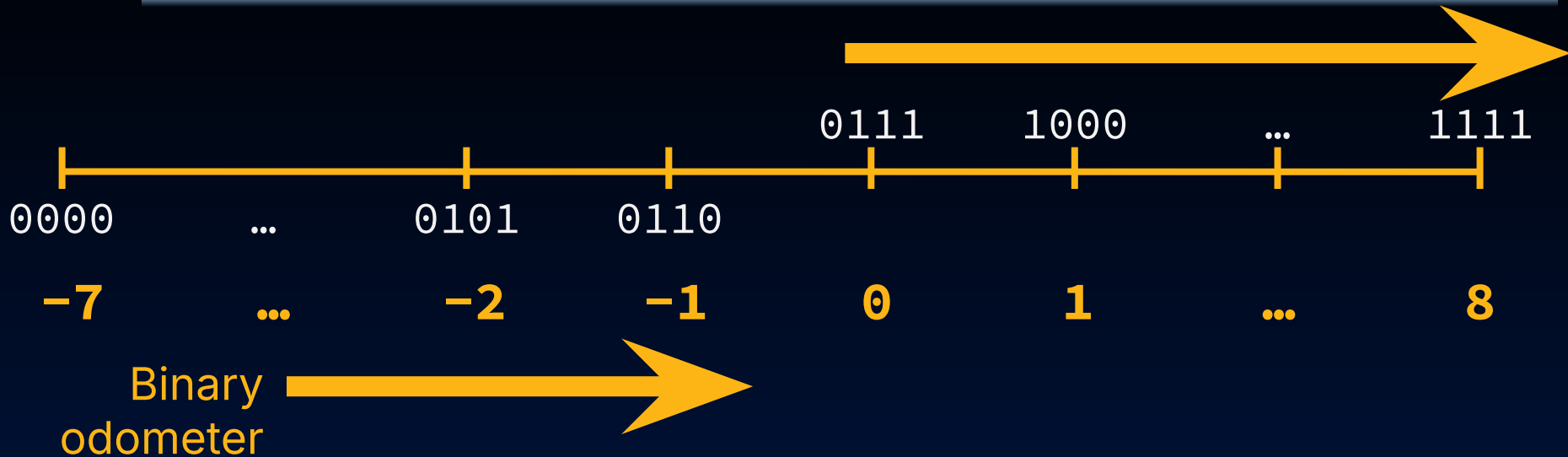
Bias Encoding

Think of an electrical signal from 0v to 15v.
How to center on 0?

- We have a system that can represent this:

- We want to represent this:

- **Bias encoding:** "Shift" the numbers so that they center on zero
- Formally:
 - Define a "bias"
 - To interpret stored binary: Read the data as an unsigned number, then **add the bias**
 - To store a data value: Subtract the bias, then store the resulting number as an unsigned number



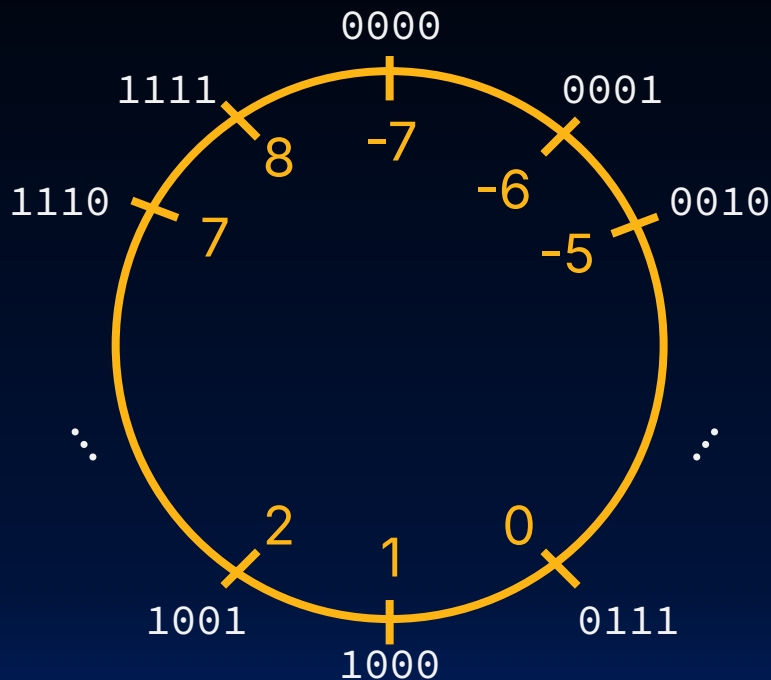
Bias Encoding



- Number = (unsigned rep) + (bias)
- With N bits, default bias is $-(2^{N-1} - 1)$
 - E.g., 4 bits, bias = $-(2^3 - 1) = -(8 - 1) = -7$
- Bias could be anything we want! (i.e., 4 bits could be #s 800-815)

Bias Encoding: $N = 4$, bias = -7

Example: $N = 4$, bias = -7



- Consider:
 - One zero
 - How many positives?
 - How many negatives?

Amazing Illustrations by Ketrina (Yim) Thompson

Binary Representations for Integers

In the early days of computing, designers made computers express numbers using **unsigned binary**. And they were content...

Until there were negative numbers.

To include negative numbers, designers came up with **sign magnitude**. That took care of the negative numbers...

But the computer had to count backwards for the negative numbers.

Plus, this introduced positive and negative zero.

Then designers created **one's complement**. Now computers only had to count in one direction.

But there were still two zeroes!

Finally, designers developed **two's complement**. Now, there was only one zero.

And they were content.

Comparing Integer Representations

Negatives and Zeros

Here, we'll choose who gets to be the world's standard for computer integers! For fun, let's introduce our contestants!

Unsigned, Sign Magnitude, One's Complement, Two's Complement, Bias

In this competition, we'll use 8-bit numbers. Now let's get started!

Round 1 - Negation

Round 1 is easy. Just tell me how you negate a number!

Unsigned: Flip sign bit

Sign Magnitude: Flip all bits

One's Complement: Flip all bits and ones

Two's Complement: Subtract from 255

Bias: Subtract from 127

Round 2 - Zeros

Now for Round 2! Show me all the ways you represent zero!

Unsigned: 0000 0000

Sign Magnitude: 0000 0000, 1000 0000

One's Complement: 0000 0000, 1111 1111

Two's Complement: 0000 0000, 1111 1111

Bias: 0111 1111

Now things are getting interesting! Unsigned and Two's Complement get two points each for having one zero and being able to represent zero with all zero bits. Bias's zero isn't all zero bits, but it gets a point for having only one zero. And though they have two zeros, Sign Magnitude and One's Complement get a point for having a zero of all zero bits.

Comparing Integer Representations

Increments and Monotonicity

Round 3 - Incrementing

On to Round 3! Using this board, graph how your value changes when you increment your bit pattern from 00000000 to 11111111. We'll give each graph a point for having a continuous graph and a point for a constant unit slope.

Unsigned: Very nice! Unsigned has a graph that is continuous and has a unit slope. This means we can use an unsigned comparator to compare integers! We'll give Unsigned two points for that.

Sign Magnitude: What's that? Sign Magnitude has a very unusual increment indeed. It has a unit slope for positive integers, but the slope becomes -1 for negatives. Sorry, but no points for Sign Magnitude this round.

One's Complement: There's a discontinuity in the graph for One's Complement, but we do like how it has a constant unit slope. That's one point for One's Complement.

Two's Complement: Just like One's Complement, Two's Complement has a discontinuous graph and unit slope. So we'll give Two's Complement a point.

Bias: Another monotonically increasing graph with unit slope! You can use an unsigned comparator to compare integers here, too. Bias gets two points!

Comparing Integer Representations

The Thrilling Conclusion!

We've finally arrived at the end of our competition. Let's see what the scoreboard says!

	Negation	One Zero?	Zero's 0000 0000	Continuous?	Monotonically Increasing?
Unsigned					
Sign Magnitude	✓		✓	✓	✓
One's Complement	✓		✓	✓	✓
Two's Complement	✓		✓	✓	✓
Bias	✓		✓	✓	✓

Well, well! It appears we have a three-way tie among Unsigned, Two's Complement, and Bias! We can certainly give each of our winners a prize, though!

Unsigned, you'll be the representation for data whenever users call upon the unsigned modifier in C! I've heard that other languages use it, too, so you'll work for them as well.

unsigned char foo = 242;

00011000

Bias, you'll represent the exponent in IEEE-754 floating-point numbers! The fact that we can compare exponents with an unsigned comparator will come in handy!

2

signed char bar = -24;

11101000

And you, Two's Complement, because you can negate and have one zero that is expressed as all zero bits, you will be the representation of integers for binary computers all around the world!

And in summary...

- We represent “things” in computers as particular bit patterns:
 - With N bits, you can represent at most 2^N things.
 - Today, we discussed five different encodings for integers:
 - Unsigned integers
 - Signed integers:
 - Sign-Magnitude
 - Ones' Complement
 - Two's Complement
 - Bias Encoding
 - Computer architects make design decisions to make HW simple
 - Unsigned and Two's complement are C standard. Learn them!!
 - Integer overflow: The result of an arithmetic operation is outside the representable range of integers.
 - Numbers have infinite digits, but computers have finite precision.
- This can lead to arithmetic errors. More later!

**For you to consider:
How could we represent -12.75?**



L02b How best to represent -12.75? (explain shifting binary point)

2s Complement (but shift binary point)

Bias (but shift binary point)

Combination of 2 encodings

Combination of 3 encodings

We can't



L02b How best to represent -12.75? (explain shifting binary point)

2s Complement (but shift binary point)

Bias (but shift binary point)

Combination of 2 encodings

Combination of 3 encodings

We can't



L02b How best to represent -12.75? (explain shifting binary point)

2s Complement (but shift binary point)

Bias (but shift binary point)

Combination of 2 encodings

Combination of 3 encodings

We can't