I don't know what this is, but it was here last semester. Enjoy.

Lecture 1

Number Representation

CS 61C, Fall 2024 @ UC Berkeley

Slides credit: Dan Garcia, Justin Yokota, Peyrin Kao

Slides template credit: Josh Hug, Lisa Yan

Speedrunning this in live lecture.

You can read through this later at your own pace.

Course Logistics

Lecture 1, CS 61C, Fall 2024

Course Logistics

Big Idea: Everything is Bits

Numbers in Different Bases

- Converting Between Bases
- Commonly Used Bases

Representing Integers (Including Negatives)

- Unsigned Integers
- Sign-Magnitude
- One's Complement
- Two's Complement
- Bias Notation

Joining the Course

- Course staff **does not control getting into the course.**
  - [Contact a major advisor](#) if you have questions.

- Being added to course platforms (Ed, Gradescope, PrairieLearn) is **manual**. We do this every day. Please **do not** email us asking to be added to the platforms unless it's been more than 2 days since you joined the course.

- If you cannot currently add or waitlist the course due to declaration, fill out this form:
  - https://forms.gle/YXnooYVLQ5Zabg3u5

- If you are a concurrent enrollment student, you should receive a bCourses invitation 1−2 days after you've submitted your application.
  - It will take 1−2 days after accepting the invitation to be added to course platforms.
  - **Do not** email course staff about getting added to the course platforms unless it's been more than 2 days since you've accepted the bCourses invitation.

Course Format

Attendance will not be part of your grade.

However…

- Synchronous, in-person participation is the most effective way of learning the material.
- The next best mode is synchronous, online participation (we intend on offering at least one remote lab and discussion).
- Least ideal mode is asynchronous participation.

Communication

Ed: https://edstem.org

- Discussion forum for Q&A and announcements.
- For private matters, you can make a private post.
- We're going to restrict the use of private posts for debugging purposes this semester. Come to office hours first.

Course website: https://cs61c.org

- Course schedule, lecture slides, assigned readings, and other resources are all posted here.

Email: cs61c@berkeley.edu

- Ed response times will generally be faster, but email may be useful for private matters.
- Please don't email individual staff members.

Course Structure: Lectures

You made it here, congrats!

- Mon, Wed, Fri, 10am–11am.
- Lecture attendance is not taken.
- Lectures will be recorded and posted.
- No plans to livestream lectures over Zoom (except the first one).

Course Schedule This Week

Labs, discussions, and office hours start next week (Sep 3).

Lab 0 will be released this week and there will be no lab sections since it is setup.

- Please post on Ed if you need help!

Course Structure: Discussions

Smaller, 1-hour long sections led by a TA to practice course content.

- You can attend any discussion section you want.
- Sections start next week (Sep 3).
- We will release discussion recordings from a previous semester.
- Schedule will be available at https://cs61c.org/fa24/calendar/.

Course Structure: Labs

2 hours long, with 30 min–1 hour conceptual mini-lecture from a TA, and remaining time office hour style Q&A.

- Only lab questions in lab!
- Lab 0 will be released this week and there will be no lab sections since it is mostly setup.
  - Please come to office hours or post on the Ed if you need help!
- Synchronous labs will start next week (Sep 3).
- Schedule will be available at https://cs61c.org/fa24/calendar/.

Course Structure: Projects

There are 4 projects.

- Project 1: snek: Get comfortable coding in C and using debugging tools!
- Project 2: CS61Classify: Code in assembly language (RISC-V)!
- Project 3: CS61CPU: Build a working CPU!
- Project 4: CS61kaChow: Make things fast!

Course Structure: Staff Support

Ed discussion forum:

- We will be providing limited private Ed support this semester.
- However, public posts (and discussion) are still highly encouraged!
- **We will only answer private Ed posts if you go to office hours and a staff member told you to make an Ed post.**
  - Except for Lab 0, you can make private posts for that.
- Go to office hours! It's so much better in terms of collaboration.

Note: We usually provide increased support for projects (in the form of increased staffing) leading up to the posted, original deadline.

- As a result, continuing to work on a project after the original deadline **may lead to slower response times on Ed/longer wait times in office hours**.
- Staff will actively prioritize assignments whose deadlines have not passed.

Course Structure: Exams

Dates:

- Midterm: Monday, October 14, 8–10pm PT.
- Final: Monday, December 16, 8–11am PT.

Policies:

- All exams will be in person.
- We will have a single alternate exam, **only if you have a documented conflict**.
    - Alternate midterm: Time TBD.
    - Alternate final exam: Monday, December 16, 11am–2pm PT.
      (You can have a few minutes to walk between exam rooms.)
- Clobber: If you score better on the final exam, we will use the z-score of your final exam score to replace your midterm score.
- Curve: If the exam mean is too low, we will decrease the max score until the average is 65%. (More details on the policies page.)

Grading Breakdown

Homework: 10%

- Try as many times as you want, and get instant feedback as you work.
- Due on Tuesdays every week.

Labs: 10%

- Hands-on experience with course material.
- Usually due on Thursdays, with some exceptions.
- You can attend lab sections to work through the lab, or do them on your own.

Projects: 40%

- Apply course concepts on larger codebases. Work solo, or with a partner.
- Due on Thursdays.

Midterm: 16%

Final: 24%

Class Policies: Collaboration

Collaboration on assignments:

- Labs and homework: can work in small groups, but must turn in your OWN work.
- Projects can either be completed solo or with a partner. If you work with a partner, all work must be completed synchronously.
  - One partner getting help in OH is fine.

Limits of collaboration:

- Don't share solutions or approaches with each other (except project partners).
- Golden rule: You should never **see or have possession** of anyone else's solutions—including from past semesters.
  - We check intermediate student work.
  - Plagiarism at *any point* in your work is unacceptable.
- Be responsible and take precautions (e.g. don't give someone else access to your computer for any reason).

Class Policies: Academic Misconduct

BOTH the sender and the receiver of work will be penalized.

Out of ~600 students who took 61C last spring, ~100 students were caught.

- 1/6 of students in the class were caught doing academic misconduct.

Class Policies: Academic Misconduct

We're here to help! There are plenty of staff and resources available for you.

- You can always talk to us if you're feeling stressed or tempted to cheat.

Academic misconduct policies:

- At minimum, the student will receive negative points on the assignment.
- Example: If a project is worth 30 points, the student will receive a score of -30 on the project.
- The student will be referred to the Center for Student Conduct
- If you take the class with integrity, you don't need to worry about these!

Choose your partners wisely!

- If your partner cheats, you will be flagged as well on that assignment!

Stress Management, Mental Health and DSP

We want to reduce your stress where we can.

- Your well-being is more important than this course.

If you have a DSP letter, please submit it through the DSP portal (AIM) as soon as possible.

Extensions:

- We understand that life happens. You can request an extension via https://gradar.cs61c.org.
- We're here to support you — if we see that you're falling behind, we may offer to schedule a meeting with you.

Stress Management and Mental Health

We want to reduce your stress where we can.

Your well-being is more important than this course.

If you feel overwhelmed, there are options:

- Academically: Ask on Ed, talk to staff in office hours, set up a meeting with staff to make a plan for your success this semester.
- Non-academic: Counseling and Psychological Services (CAPS) has multiple free, confidential services.
  - Casual consultations: uhs.berkeley.edu/counseling/lets-talk
  - Crisis management: uhs.berkeley.edu/counseling/urgent
  - Check out UHS's resources: uhs.berkeley.edu/health-topics/mental-health

Stress Management and Mental Health

Life happens.

- And sometimes life is really sucky. And that's okay.
- If you find yourself in a tough situation at any point in the semester, please don't hesitate to reach out! We will not judge you in any way; our top priority is just making sure you're supported.
- We're often our own harshest critics; if you're struggling or are faced with a situation that's affecting your well-being in any way, and can't decide whether to reach out or not, please do.
- There are many ways to handle tough situations from a course-side perspective, ones that don't involve you having to fail a course or perform less than you believe you can!

Class Policies: DSP

Disabled Students' Program (DSP):

- There's a variety of accommodations UC Berkeley can help us set up for you in this class.
- https://dsp.berkeley.edu/
- Are you facing barriers in school due to a disability? Apply to DSP!
- Only instructors, logistics TAs, and student support TAs can access any DSP-related info.

Our goal is to teach you the material in our course. The more accessible we can make it, the better.

If you have a DSP letter, please submit it through the DSP portal (AIM) as soon as possible.

Big Idea: Everything is Bits

Lecture 1, CS 61C, Fall 2024

Big Idea: Bits Can Represent Anything

All data in your computer is stored as **bits**, sequences of 0s and 1s.

- High voltage wire = 1. Low voltage wire = 0.

Everything can be represented as bits.

- Numbers: 153 could be `10011001`.
- Characters: Q could be `01010001`.
- Characters: ⊞ could be `111001111001010010110000`.
- Logical values: True could be 1.
- Colors: Red could be `111111110000000000000000`.
- Pictures.
- And so on...

We should all agree on the representation.

- If you think True is 1, and I think True is `0`, we're in trouble.

Representing Things as Bits

If we have N bits, we can represent 2^N different things.

- Example: If we have 12 bits, we can represent 2^{12} = 4096 different things.

Why?

- Each bit can be 0 or 1, i.e. 2 possibilities.
- There are $\underbrace{2 \times 2 \times 2 \times ... \times 2}_{N \text{ times}}$ = 2^N different N-bit sequences.
- Each bit sequence can represent a different thing.

Powers of 2. You will memorize them from using them over and over in this class.

2^1	2
2^2	4
2^3	8
2^4	16
2^5	32
2^6	64
2^7	128
2^8	256
2^9	512
2^{10}	1024
2^{11}	2048
2^{12}	4096

Representing Things as Bits

Today's goal: Representing integers as bits.

- Representing real numbers (e.g. 5.7) as bits? Come back next week.
- Representing code as bits? Come back in a few weeks.

To represent integers as bits, we'll first take a detour and think about number bases...

Numbers in Different Bases

Lecture 1, CS 61C, Fall 2024

Decimal System (Base 10)

The base-10 digits: 0 1 2 3 4 5 6 7 8 9

Larger numbers use multiple digits.

$$4576_{10} = (4 \times 10^3) + (5 \times 10^2) + (7 \times 10^1) + (6 \times 10^0)$$

Base-10.

Octal System (Base 8)

The base-8 digits: 0 1 2 3 4 5 6 7

Maybe if we had 8 fingers instead of 10.

Larger numbers use multiple digits.

$$4576_8 \quad = (4 \times \mathbf{8}^3) + (5 \times \mathbf{8}^2) + (7 \times \mathbf{8}^1) + (6 \times \mathbf{8}^0)$$

$$= 2430_{10}$$

Base-8.

To convert bases to base-10: Write out the powers of that base (e.g. powers of 8).

Hexadecimal System (Base 16)

The base-16 digits: 0 1 2 3 4 5 6 7 8 9 A B C D E F

(10) (11) (12) (13) (14) (15)

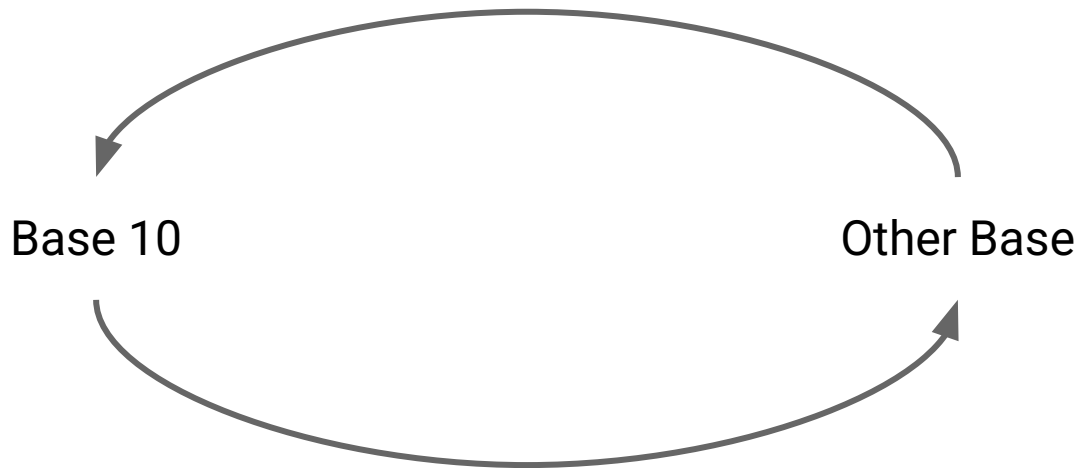We use letters if we need more digits.

Larger numbers use multiple digits.

$$4AC2_{16} = (4 \times \mathbf{16}^3) + (10 \times \mathbf{16}^2) + (12 \times \mathbf{16}^1) + (2 \times \mathbf{16}^0)$$

$$= 19138_{10}$$

Base-16.

To convert bases to base-10: Write out the powers of that base (e.g. powers of 16).

Write out powers of the base, e.g.

$(10 \times \mathbf{16}^2) + (12 \times \mathbf{16}^1) + (2 \times \mathbf{16}^0)$

Base 10

Other Base

Use the "leftover algorithm." — Not an official name.

Let's see it in action.

"Leftover Algorithm" for Converting to Base-10

Convert 73_{10} to base-4.

73 items to put in boxes.
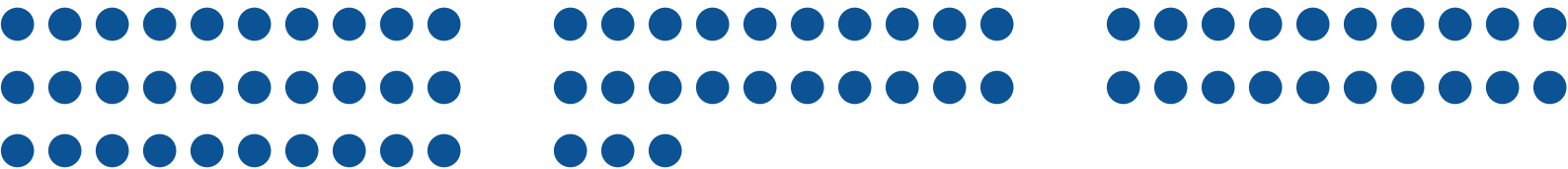
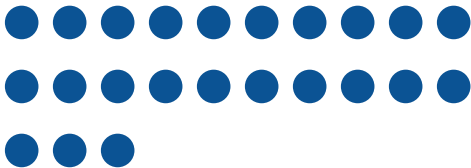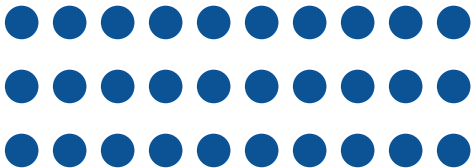| 4^5 1024s place	4^4 256s place	4^3 64s place	4^2 16s place	4^1 4s place	4^0 Ones place

The box sizes we can use:
1024, 256, 64, 16, 4, 1.

Write how many boxes
of each size we use.

Rules:

If you use a box, you have to fill it up.

Use as few boxes as possible.

"Leftover Algorithm" for Converting to Base-10

Convert 73_{10} to base-4.

4^5 1024s place	4^4 256s place	4^3 64s place	4^2 16s place	4^1 4s place	4^0 Ones place
0					

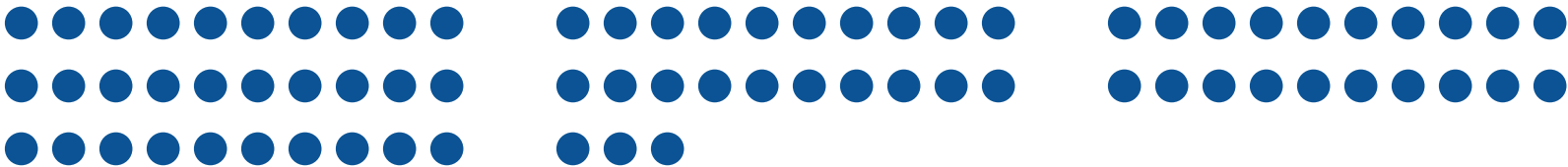How many size-1024 boxes should we use?

0. (Remember, all boxes must be full.)

Still 73 items left.

"Leftover Algorithm" for Converting to Base-10

Convert 73_{10} to base-4.

4^5 1024s place	4^4 256s place	4^3 64s place	4^2 16s place	4^1 4s place	4^0 Ones place
0	**0**				

How many size-256 boxes should we use?

0.

Still 73 items left.

"Leftover Algorithm" for Converting to Base-10

Convert 73_{10} to base-4.

● ● ● ● ● ● ● ● ● ●

Box of 64

4^5 1024s place	4^4 256s place	4^3 64s place	4^2 16s place	4^1 4s place	4^0 Ones place
0	0	**1**			

How many size-64
boxes should we use?

1. Fits 64 items.

73 − 64 = 9 items left.

"Leftover Algorithm" for Converting to Base-10

Convert 73_{10} to base-4.

● ● ● ● ● ● ● ● ●

Box of 64

4^5 1024s place	4^4 256s place	4^3 64s place	4^2 16s place	4^1 4s place	4^0 Ones place
0	0	1	**0**		

How many size-16 boxes should we use?

0.

Still 9 items left.

"Leftover Algorithm" for Converting to Base-10

Convert 73_{10} to base-4.

-

4^5 1024s place	4^4 256s place	4^3 64s place	4^2 16s place	4^1 4s place	4^0 Ones place
0	0	1	0	**2**	

Box of 4

Box of 64

Box of 4

How many size-4 boxes should we use?

2.

9 − (2 × 4) = 1 item left.

"Leftover Algorithm" for Converting to Base-10

Convert 73_{10} to base-4.

| | | Box of 64 | | Box of 4 | |
| | | | | Box of 4 | Box of 1 |

4^5 1024s place	4^4 256s place	4^3 64s place	4^2 16s place	4^1 4s place	4^0 Ones place
0	0	1	0	2	**1**

How many size-1 boxes should we use?

1.

1 − 1 = 0 items left!

"Leftover Algorithm" for Converting to Base-10

Convert 73_{10} to base-4.

4^5 1024s place	4^4 256s place	4^3 64s place	4^2 16s place	4^1 4s place	4^0 Ones place
		Box of 64		Box of 4 / Box of 4	Box of 1
0	0	1	0	2	1

0 items left. We're done!

73_{10} = 001021_4 (or just 1021_4).

"Leftover Algorithm" for Converting to Base-10

The "leftover algorithm" converts from other bases to base-10.

- Check the powers of the base. For base-4: 256, 64, 16, 4, 1.
- How many multiples of 64 fit in my number (73)?
  - 1.
  - 73 − 64 = 9 left over.
- How many multiples of 16 fit in my remaining number (9)?
  - 0.
  - Still 9 left over.
- How many multiples of 4 fit in my remaining number (9)?
  - 2.
  - 9 − (2 × 4) = 1 left over.
- How many multiples of 1 fit in my remaining number (1)?
  - 1.
  - 1 − 1 = 0 left over, which means we're done!

Commonly Used Bases

Lecture 1, CS 61C, Fall 2024

Commonly Used Bases

Some commonly used bases in computer science:

- **Base 10 (decimal)**:
  - Digits: 0 1 2 3 4 5 6 7 8 9
  - Understandable by humans.
- **Base 2 (binary)**:
  - Digits: 0 1
  - Converting numbers to base 2 lets us represent numbers as bits!
- **Base 16 (hexadecimal)**:
  - Digits: 0 1 2 3 4 5 6 7 8 9 A B C D E F
  - A convenient shorthand for writing long sequences of bits.

Writing numbers is ambiguous when we have different bases.

- 1011   Is this base 2? Base 10? Base 16? Base 7?

Disambiguation rules for this class:
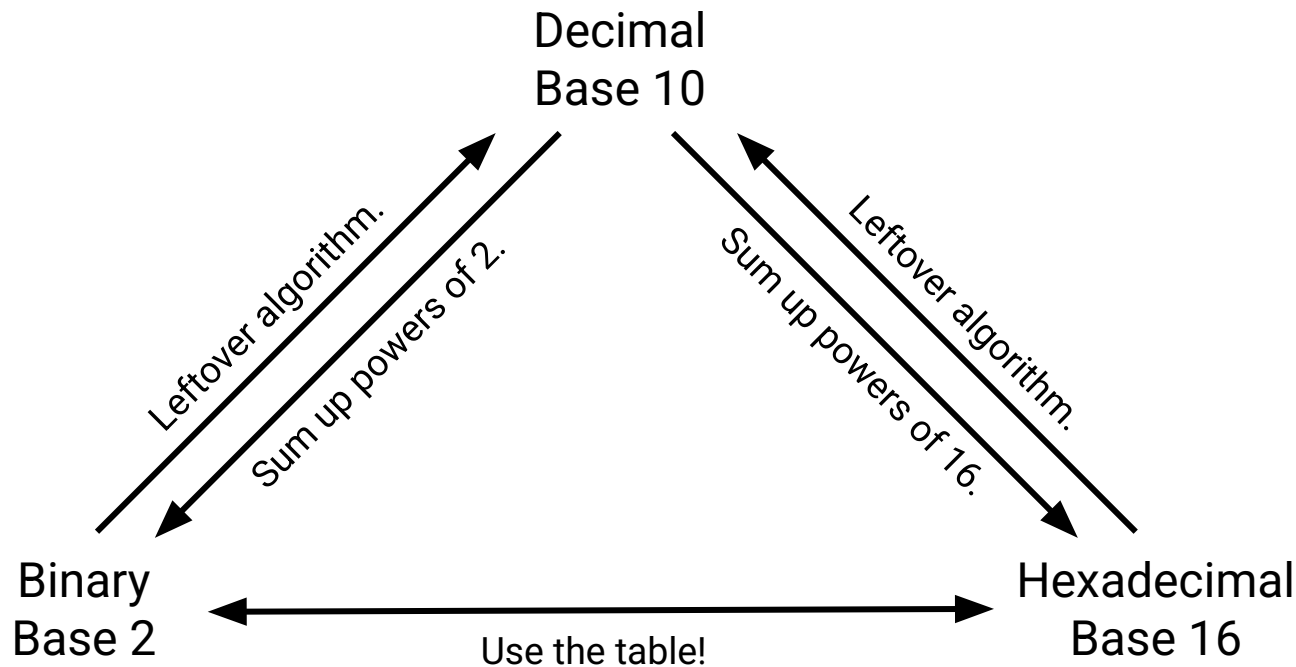
- 1011   No prefix = assume decimal.
- 0b1011   Put "0b" in front of all binary numbers.
- 0x1011   Put "0x" in front of all hexadecimal numbers.
- 01011   In C, leading zero = treat the number as base-8.   ← Weird, but true. We basically never use this in 61C.
- 1011_7 For all other bases, write the base after the number.

We sometimes add spaces for clarity: 0b1011 1001 1010 1111

If we can't figure out what base you're using on an exam, we will be sad and take off points.

Converting Between Commonly-Used Bases

Decimal
Base 10

Leftover algorithm.

Sum up powers of 2.

Sum up powers of 16.

Leftover algorithm.

Binary
Base 2

Hexadecimal
Base 16

Use the table!

Converting Between Hexadecimal and Binary – Use the Table

Each 4-bit sequence corresponds to one hex digit!

Why?

- How many 4-bit sequences? 2^4 = 16.
- How many hexadecimal digits? 16.

To convert between hex and binary, just use the table!

Hexadecimal is a convenient shorthand for long sequences of bits.

- Ugly: 0b1110011110010100101110000
- Nice: 0xE794B0

Decimal	Binary	Hex
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Converting Between Hexadecimal and Binary – Use the Table

Beware of padding with zeros!

- Left-padding is okay. $52_{10} = 0052_{10}$.
- Right-padding is bad. $52_{10} \neq 5200_{10}$.

Binary → hex: Left-pad if needed.

- Example: Convert `0b110010` to hex.
- Correct: `0b00110010` is `0x32`.
- Wrong: `0b11001000` is *not* `0xC8`.

Hex → binary: Drop leading zeroes if needed.

- Example: Convert `0x1D` to binary.
- `0x1D` = `0b00011101` = `0b11101`.

Decimal	Binary	Hex
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Units of Measurement

Groups of bits have special names.

- 1 **byte** = 8 bits.   2 hex digits. 2^8 = 256 different values.
- 1 **nibble** = 4 bits.   1 hex digit. 2^4 = 16 different values.

Or "nybble" if you want to be cute.

Unsigned Integers

Lecture 1, CS 61C, Fall 2024

Unsigned Integers: Properties

We now have a way to represent numbers as bits!

- Just convert from base-10 to base-2.

Resulting data type: **N-bit unsigned integer**.

- Can represent 2^N different numbers.
- Smallest number: `0b0000...000` represents 0.
- Largest number: `0b1111...111` represents $2^N - 1$.

N = 32 is common in computers. Can represent $[0, 2^{32} - 1]$, roughly 4 billion numbers.

Useful if you don't need negative numbers.

- Example: Variable storing the length of a list.

Math in base-10: Just like you learned in elementary school.

$$
\begin{array}{r}
{}^{1}{}^{1}{}^{1} \\
5\quad 0\quad 9\quad 8 \\
+\quad 6\quad 8\quad 1\quad 7 \\
\hline
1\quad 1\quad 9\quad 1\quad 5
\end{array}
$$

Math in base-2 is basically the same.

```
      1     1     1
      1  1  1  1
+     0  1  1  0
─────────────────
   1  0  1  0  1
```

In binary:

- 1 + 1 = 10.   Write 0, carry the 1.
- 1 + 1 + 1 = 11.   Write 1, carry the 1.

Unsigned Integers: Number Line View

Overflow occurs when we exceed the largest value and wrap back around to 0.

- Example in 8-bit binary: 11111111 + 00000001 = 00000000.

Negative overflow occurs when we try to go below 0 and wrap around to large values.

- Example in 8-bit binary: 00000001 − 00000010 = 11111111.
- Don't call it underflow. That's a different thing.

The odometer: Counting upwards in base-2.

| 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 1001 | 1010 | 1011 | 1100 | 1101 | 1110 | 1111 |

Overflow!

Unsigned Integers: Pros and Cons

Some desirable properties of number representation systems:

Unsigned Integer	
Can represent negative numbers.	✘
Doing math is easy.	✔
Every bit sequence represents a unique number.	✔

Sign-Magnitude

Lecture 1, CS 61C, Fall 2024

Sign-Magnitude: Properties

Idea: Use the left-most bit to indicate if the number is positive (0) or negative (1).

This is called **sign-magnitude** representation.

- Smallest number: 0b1111...111 represents $-(2^{N-1} - 1)$.
- Largest number: 0b0111...111 represents $+(2^{N-1} - 1)$.
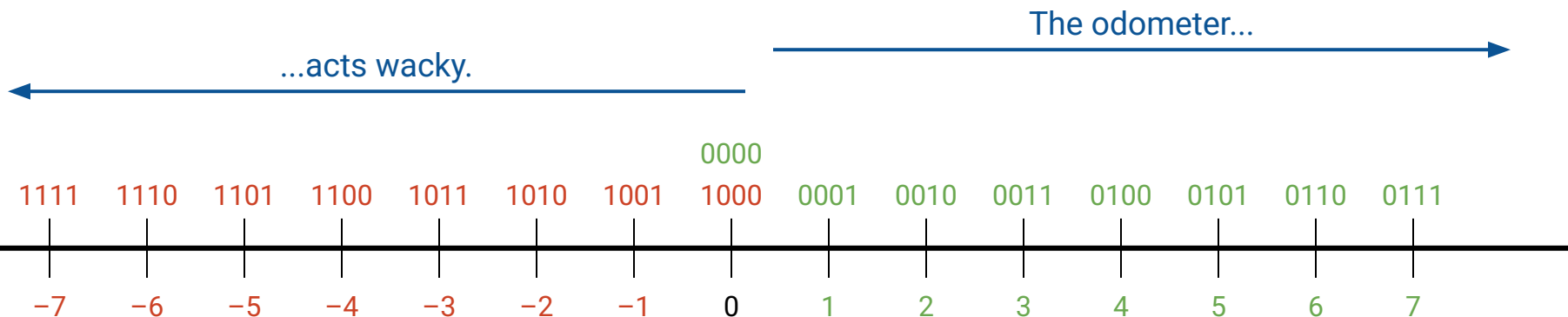
Notice: There's no free lunch.

- In exchange for representing some negative numbers...
- ...we can't represent as many positive numbers.
- We often like to represent an equal number of positive and negative numbers.

Sign-Magnitude: Number Line View

The number line is wacky in sign-magnitude!

- If we count upwards in base-2, the resulting numbers increase...
- ...then they start decreasing?!

The odometer...

...acts wacky.

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

Sign-Magnitude: Pros and Cons

Sign-Magnitude	
Can represent negative numbers.	✔
Doing math is easy.	✘
Every bit sequence represents a unique number.	✘

Number line was wacky.

Two representations of zero.

Problems:

- Imagine writing code, or building a circuit, to add sign-magnitude numbers.
  - Elementary-school addition doesn't work anymore.
  - "If sign bits are 1 and 0, subtract. Else, if sign bits are 0 and 1..."
- 0b1000...000 and 0b0000...000 both represent zero!

Sign-magnitude is rarely used as-is in practice.

One's Complement

Lecture 1, CS 61C, Fall 2024

Idea: If the number is negative, flip the bits.

- +7 is `0b00111`.
- −7 is `0b11000`.
- Left-most bit acts like a sign bit.
  If it's 1, someone flipped the bits, so number must be negative.

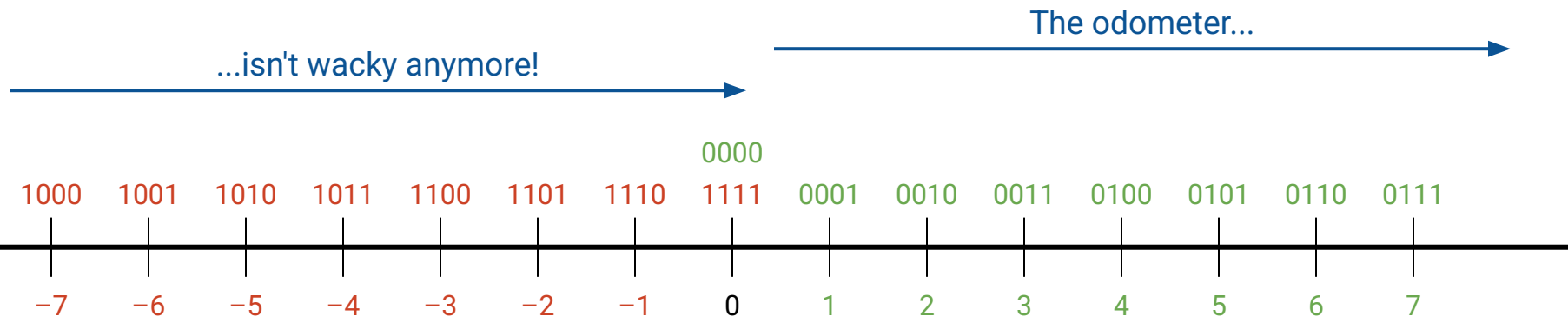This is called **one's complement** representation.

- Smallest number: `0b1000...000` represents $-(2^{N-1} - 1)$.
- Largest number: `0b0111...111` represents $+(2^{N-1} - 1)$.

One's Complement: Number Line View

The number line is no longer wacky in one's complement.

- If we count upwards in base-2, the resulting numbers are always increasing.
- There's an unavoidable overflow, just like in unsigned.
  Most positive number + 1 = Most negative number.

...isn't wacky anymore!

The odometer...

| 1000 | 1001 | 1010 | 1011 | 1100 | 1101 | 1110 | 1111 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 |

| −7 | −6 | −5 | −4 | −3 | −2 | −1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

One's Complement: Pros and Cons

One's Complement	
Can represent negative numbers.	✔
Doing math is easy.	✔
Every bit sequence represents a unique number.	✘ Two representations of zero.

Solved some of our problems:

- Counting on the number line isn't wacky anymore.
- But `0b1111...111` and `0b0000...000` both represent zero!

One's complement is not used in practice, but it leads to our next solution...

Two's Complement

Lecture 1, CS 61C, Fall 2024

From One's Complement to Two's Complement

Our remaining problem: We have two representations of zero.

0000							
1000	1001	1010	1011	1100	1101	1110	1111	0001	0010	0011	0100	0101	0110	0111

−7 −6 −5 −4 −3 −2 −1 0 1 2 3 4 5 6 7

To fix it, we can shift all the negative representations over by one.

1000 1001 1010 1011 1100 1101 1110 1111 0000 0001 0010 0011 0100 0101 0110 0111

−8 −7 −6 −5 −4 −3 −2 −1 0 1 2 3 4 5 6 7

Idea: If the number is negative, flip the bits, *and add one.*

Because we shifted
to avoid double-zero.

This is called **two's complement** representation.

- Smallest number: 0b1000...000 represents $-(2^{N-1})$.
- Largest number: 0b0111...111 represents $+(2^{N-1}-1)$.

Two's Complement: Number Line View
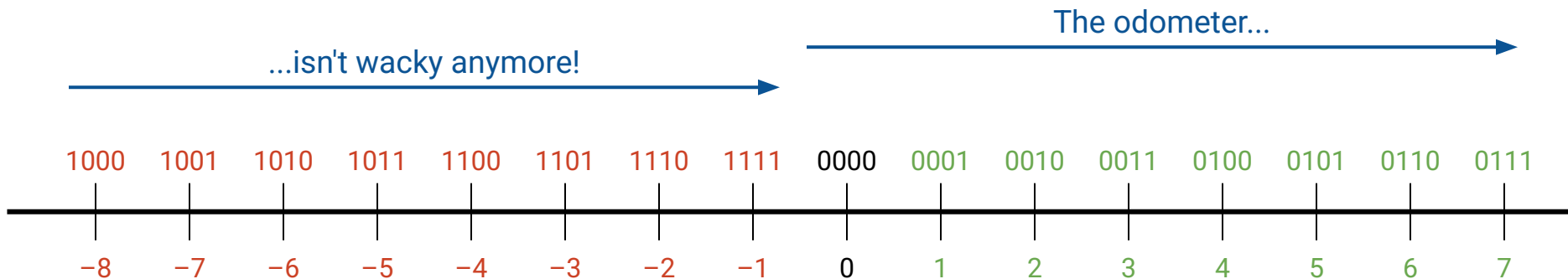
Just like its cousin, one's complement, the number line isn't wacky.

- If we count upwards in base-2, the resulting numbers are always increasing.

...isn't wacky anymore!

The odometer...

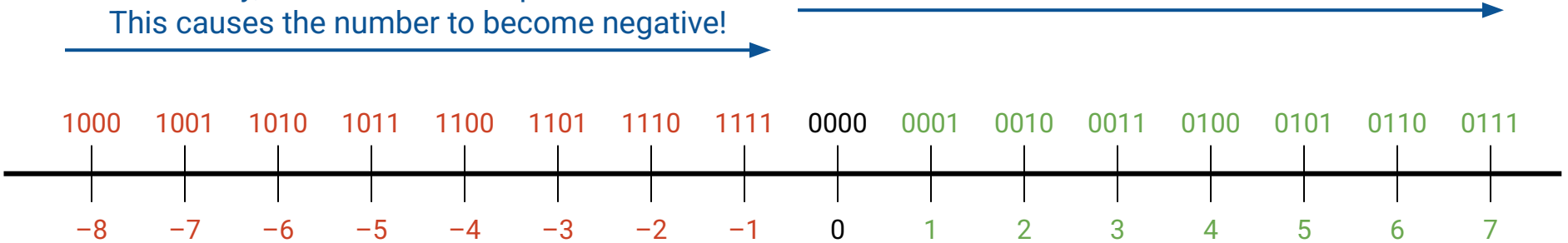| 1000	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111
−8	−7	−6	−5	−4	−3	−2	−1	0	1	2	3	4	5	6	7

Two's Complement: Properties

Another definition: The left-most power of 2 is now negative, not positive.

- Left-most bit 0: Read the rest of the number as an unsigned integer.
- Left-most bit 1: Subtract a big power of 2. Resulting number is negative!

-2^5 -32s place	2^4 16s place	2^3 8s place	2^2 4s place	2^1 2s place	2^0 Ones place
1	0	1	0	0	1

Eventually, the left-most bit flips from 0 to 1.
This causes the number to become negative!

The odometer counts...

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

Two's Complement: Conversion Algorithm

To convert two's complement to a signed decimal number:

- If left-most digit is 0: Positive number.
  - Just read it as unsigned.
- If left-most digit is 1: Negative number.
  - Flip the bits, and add 1.
  - Convert to base-10, and stick a negative sign in front.

Example: What is `0b1110 1100` in decimal?

- Flip the bits: `0b0001 0011`
- Add one: `0b0001 0100`
- In base-10: -20

Two's Complement: Conversion Algorithm

To convert a signed decimal number to two's complement:

- If number is positive:
  - Just convert it to base-2.
- If number is negative:
  - Pretend it's unsigned, and convert to base-2.
  - Flip the bits, and add 1.

Example: What is −20 in two's complement binary?

- In base-2:   `0b0001 0100`
- Flip the bits: `0b1110 1011`
- Add one: `0b1110 1100`

Two's Complement: Pros and Cons

Two's Complement	
Can represent negative numbers.	✔
Doing math is easy.	✔
Every bit sequence represents a unique number.	✔

Solves our sign-magnitude problems:

- No more double-zero! ← Because we shifted.
- Elementary-school addition works just fine. ← Because the number line isn't wacky.
  ← Deep connection to modular arithmetic.

Two's complement is the most common way to represent signed integers.

[Optional] Two's Complement: Deep Connection to Modular Arithmetic

- Because of overflow, addition behaves like modular arithmetic.
- 11 and −5 are the same number in mod land: 11 mod 16.

Bit sequence interpreted as unsigned:

2^3 8s place	2^2 4s place	2^1 2s place	2^0 Ones place	
1	0	1	1	= 11

The same sequence interpreted as two's complement:

-2^3 −8s place	2^2 4s place	2^1 2s place	2^0 Ones place	
1	0	1	1	= −5

These are always equal mod 2^N = 16.

- So, adding 11 or adding −5 gives the same answer in mod land.
- So, it doesn't matter if I read `0b1011` as 11 or −5.

1000	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111

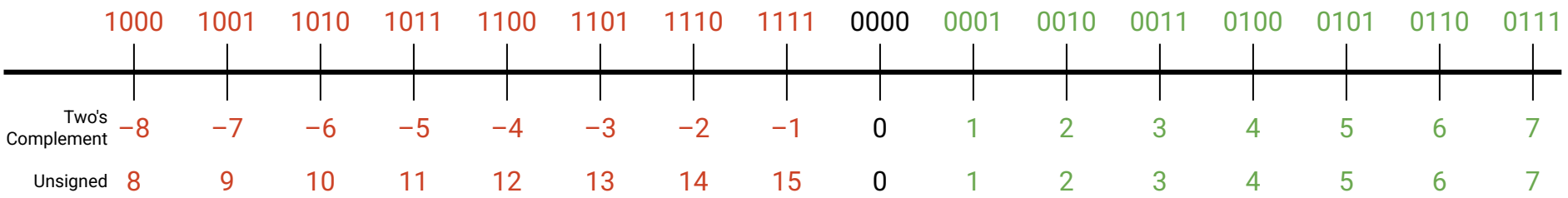| Two's Complement	−8	−7	−6	−5	−4	−3	−2	−1	0	1	2	3	4	5	6	7
Unsigned	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7

[Optional] Two's Complement: Deep Connection to Modular Arithmetic

Punchline: Unsigned math and two's complement math basically work the same.

- If an operation works in mod land, it doesn't care whether its inputs are two's complement or unsigned.
- You can add/subtract/multiply two's complement numbers as if they were unsigned.
- You'll magically* get the right answer.

*Not really magical, just mathematical. It does have a nice ring to it.

	1000	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111
Two's Complement	−8	−7	−6	−5	−4	−3	−2	−1	0	1	2	3	4	5	6	7
Unsigned	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7

Bias Notation

Lecture 1, CS 61C, Fall 2024

Why not stop at two's complement? Some oddities:

- 0b0000 doesn't represent the smallest number.
- There's an overflow in the middle of counting.

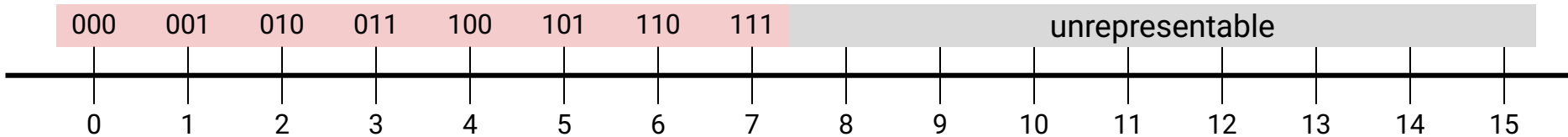The overflow in the middle of the odometer is kind of weird.

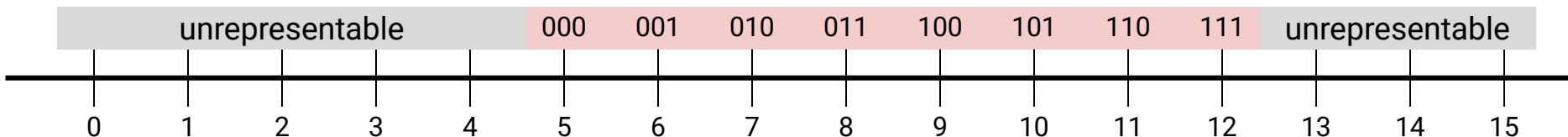| 1000	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111
−8	−7	−6	−5	−4	−3	−2	−1	0	1	2	3	4	5	6	7

What if I wanted to represent a different range of values?

- So far, we've had equal amounts of positive and negative values.
- What if I wanted to represent numbers from 5 to 12?
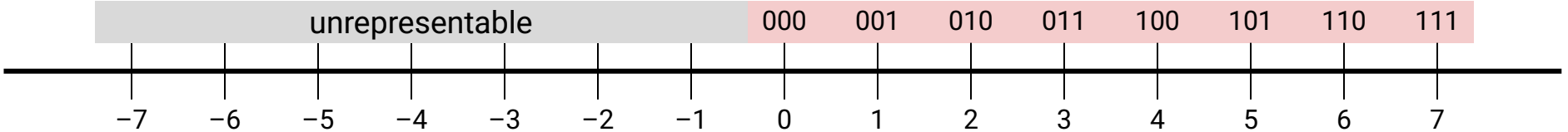
Idea: Take the unsigned numbers...

| 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 | unrepresentable |

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |

...and shift them over to the desired range!

| unrepresentable | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 | unrepresentable |

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |

Introducing Bias Notation

We can also take the unsigned numbers...

	unrepresentable				000	001	010	011	100	101	110	111

−7 −6 −5 −4 −3 −2 −1 0 1 2 3 4 5 6 7

...and shift them to center around zero.

Odometer counts up continuously!

unrepresentable		**000**	001	010	011	100	101	110	111	unrepresentable	

−7 −6 −5 −4 −3 −2 −1 0 1 2 3 4 5 6 7

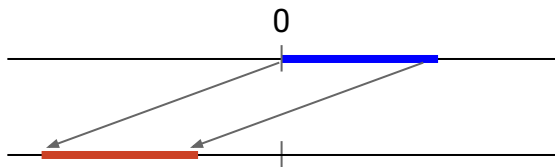0b000 represents the
smallest number!
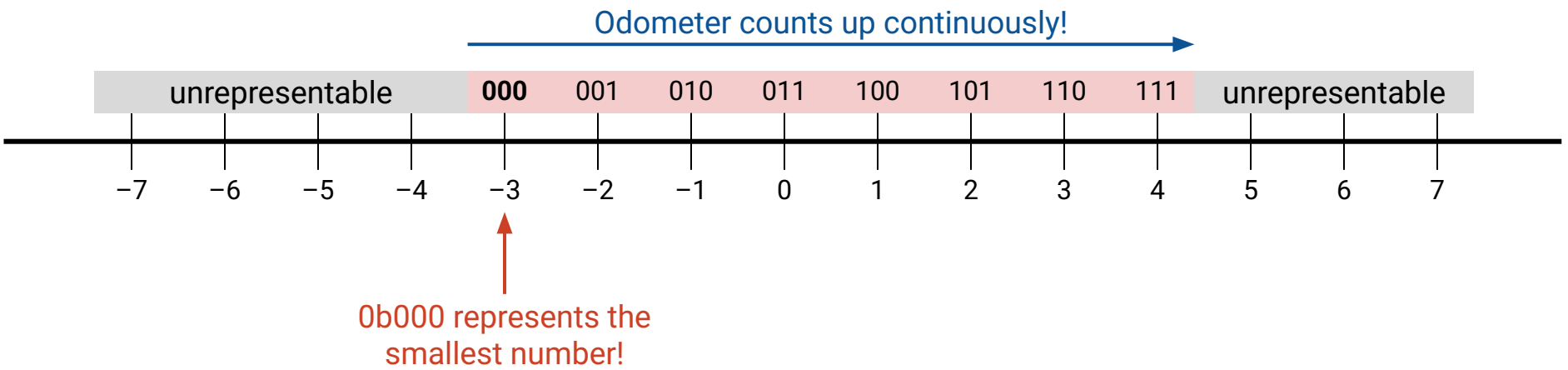
Bias Notation: Properties
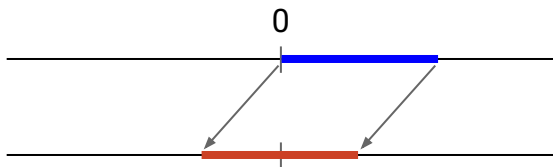
Idea: Just like unsigned, but shifted on the number line.

This is called **bias** representation.

- Smallest number: `0b0000...000` represents *bias*.
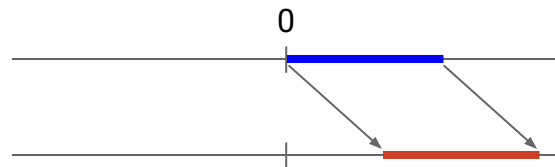- Largest number: `0b1111...111` represents $2^N - 1 + bias$.

We can pick a bias based on what numbers we want to represent.

To represent negative numbers, pick a very negative bias.

To center at 0, pick the **standard bias** of $-(2^{N-1} - 1)$.

To represent positive numbers, pick a very positive bias.

Bias Notation: Conversion Algorithm

To convert biased notation to decimal:

- Read as unsigned decimal.
- Add the bias.

Example: Assuming standard bias, what is `0b00000001` in decimal?

- N = 8, so standard bias is: $-(2^{8-1} - 1) = -127$.
- Read as unsigned: 1
- Add the bias: 1 + (--127) = −126

Intuition:

- Unsigned 8-bit number represents: [0, 255]
- Standard bias shifts to center at 0: [−127, 128]
- `0b00000000` is the smallest number (−127), so `0b00000001` must be −126.

Bias Notation: Conversion Algorithm

To convert decimal to biased notation:

- Subtract the bias.
- Convert to unsigned binary.

Example: What is −126 in 8-bit biased notation?

- $N = 8$, so standard bias is: $-(2^{8-1} - 1) = -127$.
- Subtract the bias: $-126 - (-127) = 1$
- Write in base-2: `0b00000001`

Note: In this class, if we don't specify the bias, assume the standard bias.

Yes, I know subtracting a negative bias is annoying, but that's the convention we use. Sorry.

Bias Notation: Pros and Cons

Bias Notation	
Can represent negative numbers.	✔
Doing math is easy.	✘
Every bit sequence represents a unique number.	✔

Problems:

- Math is hard. Adding `0b0000` actually adds a negative number?!
- All users have to agree on the bias to use.

Really useful if the range of values you want to represent is not centered at 0.

- Example: If we measure weather in Kelvin, temperatures are in [273, 323].

Summary: Number Representation

Everything is represented as sequences of bits in the computer.

- N bits = 2^N unique bit sequences = We can represent 2^N things.
- Hexadecimal is a useful shorthand for long bit sequences.

We converted numbers between common bases: binary, decimal, hexadecimal.

We saw different ways to represent integers as bits:

If you're here to copy this on your cheat sheet, make sure you actually understand how to derive the formulas, or else they won't help on exams.

	Handling negatives:	Easy to math?	No double zero?	Widely used?	Smallest Number:		Largest Number:	
Unsigned	N/A	✔	✔	✔	0b0000...000	0	0b1111...111	$2^N - 1$
Sign-Magnitude	Left-most bit is the sign.	✘	✘	✘	0b1111...111	$-(2^{N-1} - 1)$	0b0111...111	$+(2^{N-1} - 1)$
One's Complement	Flip the bits.	✔	✘	✘	0b1000...000	$-(2^{N-1} - 1)$	0b0111...111	$+(2^{N-1} - 1)$
Two's Complement	Flip the bits, +1.	✔	✔	✔	0b1000...000	$-(2^{N-1})$	0b0111...111	$+(2^{N-1} - 1)$
Bias Notation	Bias = $-(2^{N-1} - 1)$.	✘	✔	✔	0b0000...000	$bias$	0b1111...111	$2^N - 1 + bias$