Announcements

- Lab 00 + Welcome Form: Released! Due Tue 1/27
  - On the fence about **Regular** vs. **Bridge** vs. **Video** discussion?
    - Choose **Regular/Bridge to be assigned a time** that works for you. Video students (the default) cannot attend in-person.
    - 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

- **CONCURRENT ENROLLMENT**
  - Everyone can get in, but official roster enrollment will take longer
    - Do NOT email us at this time!
  - If by **Monday evening**, you cannot complete lab00 because you cannot sign up for an instructional account, email cs61c@berkeley.edu

CC BY NC SA

CS61C

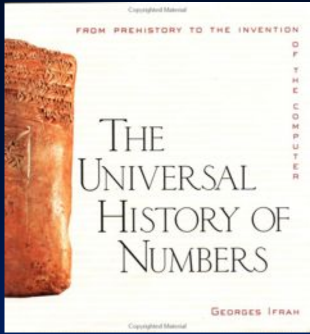
Great Ideas in Computer Architecture

(a.k.a. Machine Structures)

Assistant Teaching Professor
Lisa Yan

Number Representation

Great book ⇒
The Universal History of Numbers
by Georges Ifrah



UC Berkeley

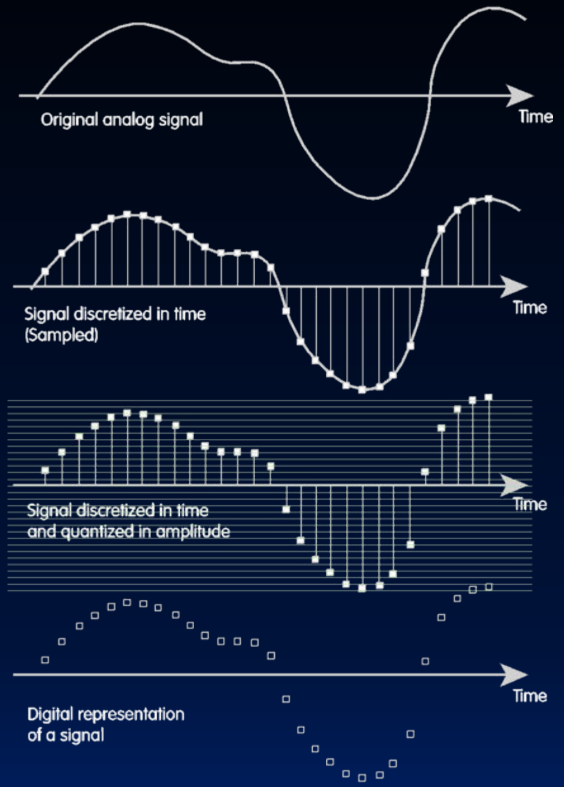
cs61c.org

CC BY NC SA

Data input: Analog → 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 tick-mark) “yardstick”, it lies.



Yan, SP26
BY NC SA

Digital data not necessarily born Analog…

hof.povray.org

BY NC SA

Discuss

1. How many "things" can be represented by 4 bits?

A. 4
B. 8
C. 16
D. 64
E. Something else

2. What does this particular 4-bit pattern represent?

`1011`

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

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

BIG IDEA: Bits can represent anything!!

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

4 bits,
each 0 or 1

____ ____ ____ ____

1011

The meaning of a bitstring depends on the program context!

Today: Many ways to use N bits to represent numbers

Yan, SP26

CC BY NC SA

Pop quiz??

1. [no pollEV, just discuss]
How many "things" can be represented by 4 bits?

A. 4
B. 8
C. 16
D. 64
E. Something else

2. [no pollEV, just discuss]
What does this particular 4-bit pattern represent?

```
1011
```

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

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

Yan, SP26
CC BY NC SA

BIG IDEA: Bits can represent anything!!

- Logical values? 1 bit
  - One possible convention: 0 → False, 1 → True
- Characters?
  - A, ..., Z: 26 letters → 5 bits (26 ≤ 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)

Yan, SP26

CC BY NC SA

Agenda

Binary, Decimal, Hex

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

BY NC SA

Number vs Numeral

Numeral

A symbol or name that stands for a number

e.g., *4* , *four* , *quatro* , *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"*

Yan, SP26

CC BY NC SA

Decimal: Base 10 (Ten) #s

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

Example:

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

Yan, SP26

BY NC SA

Base 2 (Two) #s, Binary

Digits: 0, 1 (**bi**nary digi**ts** → bits)

Example: Binary number "**1101**"

Convert to decimal:

$$\mathbf{0b1101} = \mathbf{1101}_2 = (\mathbf{1}\mathrm{x}2^3) + (\mathbf{1}\mathrm{x}2^2) + (\mathbf{0}\mathrm{x}2^1) + (\mathbf{1}\mathrm{x}2^0)$$
$$= 8 + 4 + 0 + 1$$
$$= 13$$

Common binary shorthand:
`0b1101`

Yan, SP26
CC BY NC SA

Discuss #2

1. How to represent 165 in binary?

2. Is this binary value human-readable?
   What if it appeared multiple times in a program?

0b11110100101

BY NC SA

Consider

- How to represent 165 in binary?

1	0	1	0	0	1	0	1
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1

~~165~~ ~~37~~ ~~5~~ ~~1~~ 0

Yan, SP26

CC BY NC SA

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:

0xA5 $= \mathbf{A5}_{16} = (\mathbf{10} \times 16^1) + (\mathbf{5} \times 16^0)$

$= 160 + 5$

$= 165$

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

CC BY NC SA

Hexadecimal is a Useful Shorthand for Binary (1/2)

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

Memorize this table.

Yan, SP26
BY NC SA

Hexadecimal is a Useful Shorthand for Binary (2/2)

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

The number 165

- In binary ("prefix 0b")

$$0b\underbrace{1010}_{A}\underbrace{0101}_{5}$$

$$0xA5$$

- In hexadecimal (prefix "0x")

Yan, SP26

CC BY NC SA

Back to Question 2 (1/2)

1. How to represent 165 in binary? `0b10100101`

   `= 0xA5`

2. Is this binary value human-readable?
   What if it appeared multiple times in a program?

`0b11110100101`

Yan, SP26

CC BY NC SA

Zero pad leftmost group of 4 bits as needed

1. How to represent 165 in binary? `0b10100101`

   `= 0xA5`

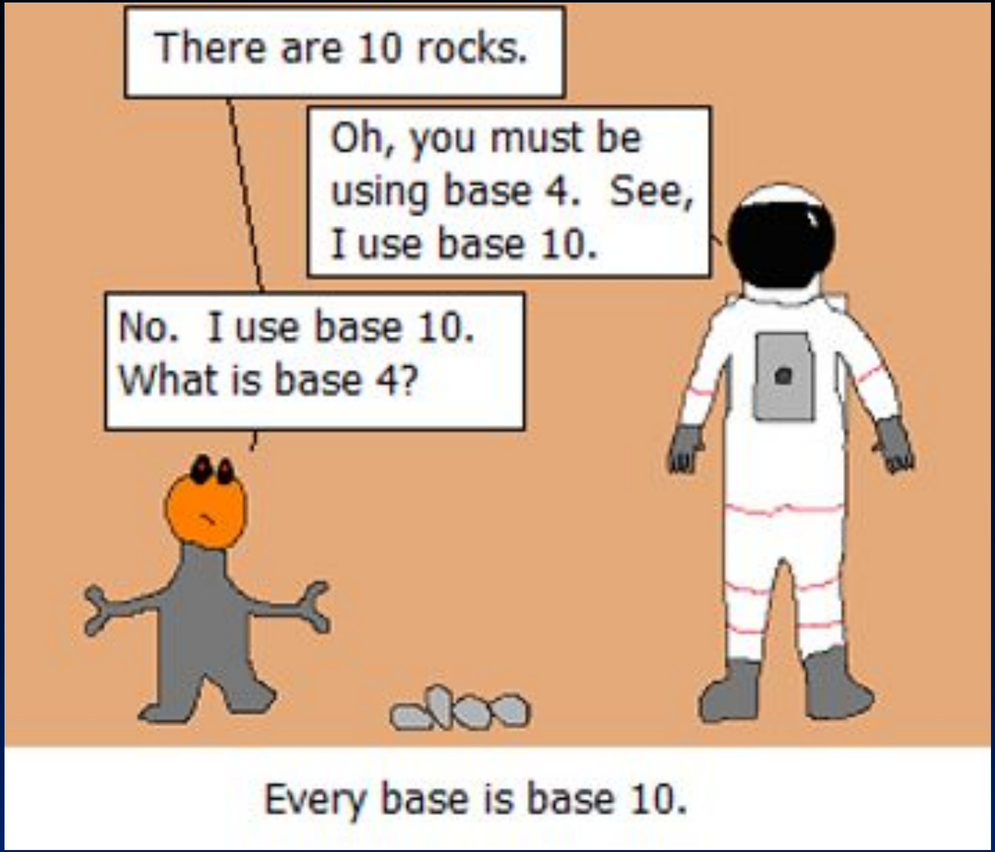
2. Is this binary value human-readable?
   What if it appeared multiple times in a program?

$$\underbrace{\texttt{0011}}_{\texttt{3}}\quad\underbrace{\texttt{1010}}_{\texttt{A}}\quad\underbrace{\texttt{0101}}_{\texttt{5}}$$

Yan, SP26

CC BY NC SA

Which base do we use? (1/2)



How many rocks?

CC BY NC SA

Which base do we use? (2/2)

- **Decimal:** great for humans, especially when doing arithmetic
- **Hex:** if human looking at long strings of binary numbers, it's 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**

- Different reps of the same number:
  - 32_{ten} == 32_{10} == `0x20` == 100000_2 == `0b100000`

Yan, SP26

CC BY NC SA

Agenda

Integer Representations

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

CC BY NC SA

How do we pick a representation for <u>integers</u>? (1/2)

- Want a representation that supports common integer operations:
    - Add them
    - Subtract them
    - Multiply them
    - Divide them
    - Compare them (<, =, ≠, ≤, etc.)

Yan, SP26

CC BY NC SA

N-Bit Unsigned Integer Representation

- Most computers use the **unsigned** integer representation from mathematics (that we just discussed.)
    - C's `uint8_t`, `uint16_t`, etc.: [0, 2^N-1]
- 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...

```
 11    carry bits
 1010
+ 0111
 10001
```


Yan, SP26

CC BY NC SA

How do we pick a representation for integers?(2/2)

- While strictly speaking, binary numerals have an infinite number of digits, **hardware has limits**.
  - Must also specify the number of bits for an integer representation.
- An N-bit representation can only represent 2^N things. When doing arithmetic, numbers might "wrap around."

Car odometer

With a 4-bit unsigned integer representation, 10 + 7 = 1.

$$
\begin{array}{r@{\quad}l}
11 & \text{carry bits} \\
1010 & \\
+\ \underline{0111} & \\
\text{dropped}\ 1\,0001 &
\end{array}
$$

Yan, SP26

CC BY NC SA

Number line vs. number "wheel"

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

Number wheels visualize integer overflow.

Yan, SP26
BY NC SA

Agenda

Sign-Magnitude, Ones' Complement

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

Let's move to discussing **signed** integer representations, which can represent positive and negative #s.

CC BY NC SA

N-Bit Signed Integer Representations

For the proposed N-bit signed integer representations, consider:

1. How to represent 5? Is it easy to convert to -5?

2. Is it easy to identify positive vs. negative from a bit pattern?

3. Is addition easy to implement? Example: 5 + -5 = 0

4. How to represent zero?

5. [at home] What is the positive integer range? Negative integer range?

Ain't no free lunch: We still have N bits.
Goal: half positive, half negative, and zero

CC BY NC SA

Two "strawman" designs for N-bit signed integer rep (1/2)

- Sign-Magnitude
  - Leftmost bit is **sign bit**
  - Remaining N-1 bits is magnitude, i.e., numerical value

$+5_{ten}$ = `0b 0000 0101`

-5_{ten} = `0b 1000 0101`

Yan, SP26

CC BY NC SA

Two "strawman" designs for N-bit signed integer rep. (2/2)

- One's Complement
  - (reserve leftmost bit for sign)
  - If positive, unsigned rep
  - If negative, **flip** bits of positive rep

$+5_{ten}$ = `0b 0000 0101`

-5_{ten} = `0b 1111 1010`

Yan, SP26

CC BY NC SA

Discuss #3

- Sign-Magnitude
  - Leftmost bit is **sign bit**
  - Remaining N-1 bits is magnitude, i.e., numerical value
- One's Complement
  - (reserve leftmost bit for sign)
  - If positive, unsigned rep
  - If negative, **flip** bits of positive rep

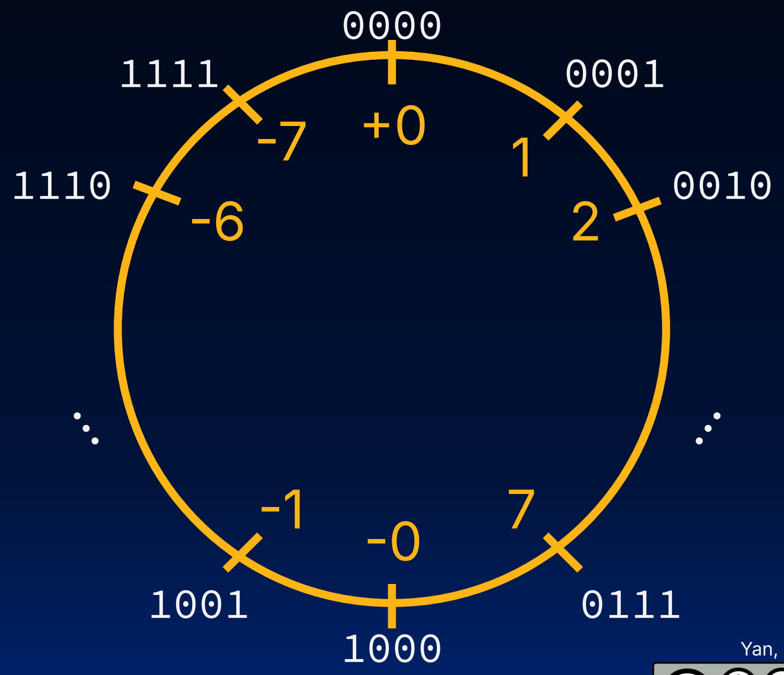
1. How to represent 5? -5? [done]
2. How to represent zero?
3. Is it easy to identify positive vs. negative?
4. Is addition easy to implement? Example: 5 + -5 = 0

Sign-Magnitude: Rarely Used

$+5_{ten}$ = 0b 0000 0101

-5_{ten} = 0b 1000 0101

- 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



Yan, SP26

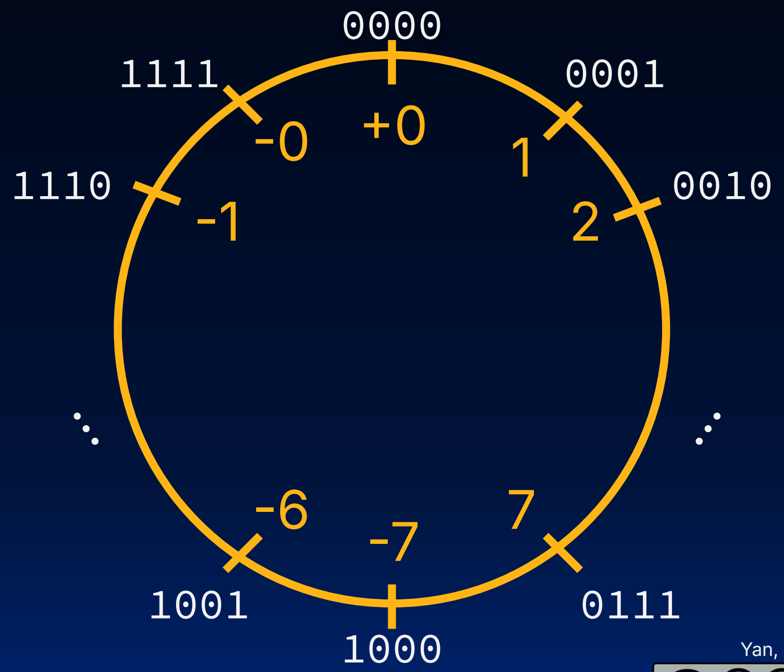
CC BY NC SA

One's Complement: Old but OK

$+5_{ten}$ = 0b 0000 0101

-5_{ten} = 0b 1111 1010

- 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
- While used for a while on some computer products
  - It's not currently used in current hardware



Yan, SP26
BY NC SA

Agenda

Two's Complement

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

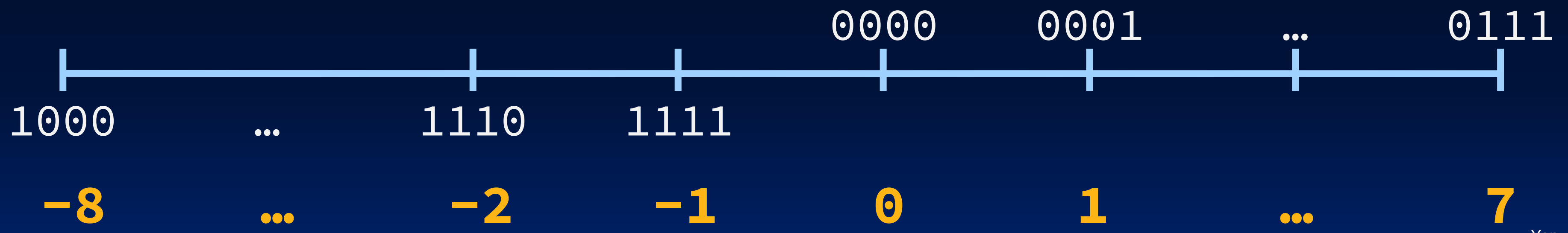
Yan, SP26

CC BY NC SA

Two's Complement: The Motivation

- Ones' complement:

- The problem: "Overlap" creates two 0s.
- The solution: **Shift the negative mappings left by one.**



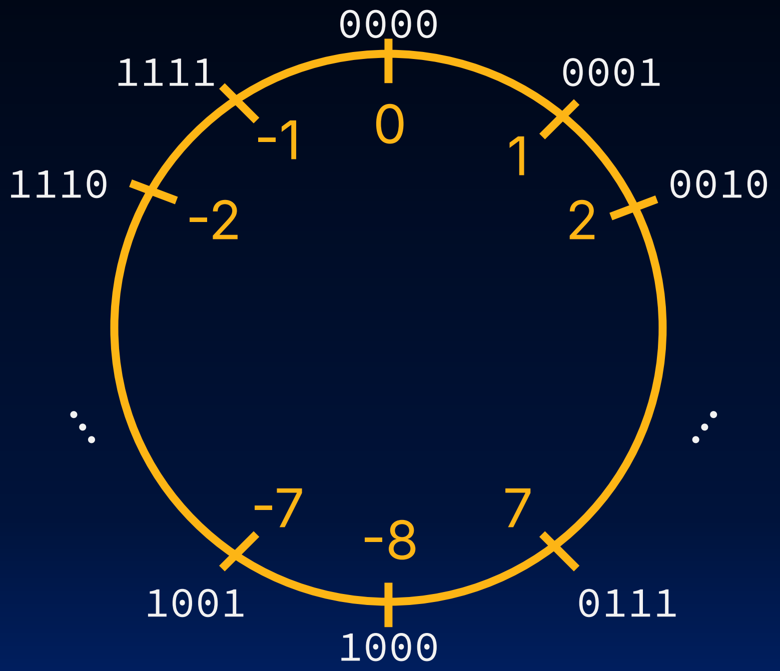
Yan, SP26
CC BY NC SA

Two's Complement: Why?

$+5_{ten}$ = `0b 0000 0101`

-5_{ten} = `0b 1111 1011`

- Advantages:
  - Leftmost bit ("**most significant bit**") is still effectively sign bit
  - One zero
  - **Simple hardware for addition**
  - Incrementing binary odometer consistent on the # line
- Two's complement is the representation used for all C23 signed integers.
  - `int8_t`, `int16_t`, etc.



Yan, SP26

CC BY NC SA

Two's Complement: Procedural Algorithm

- Two's Complement
  - If positive, unsigned rep
  - If negative:
    - Start from positive rep
    - **flip** bits
    - **add one**

5_{ten} `0101` → `1010` → `1011` -5_{ten}

- Negative → positive is same procedure: flip and add one.
  - Makes hardware easy!

-5_{ten} `1011` → `0100` → `0101` 5_{ten}

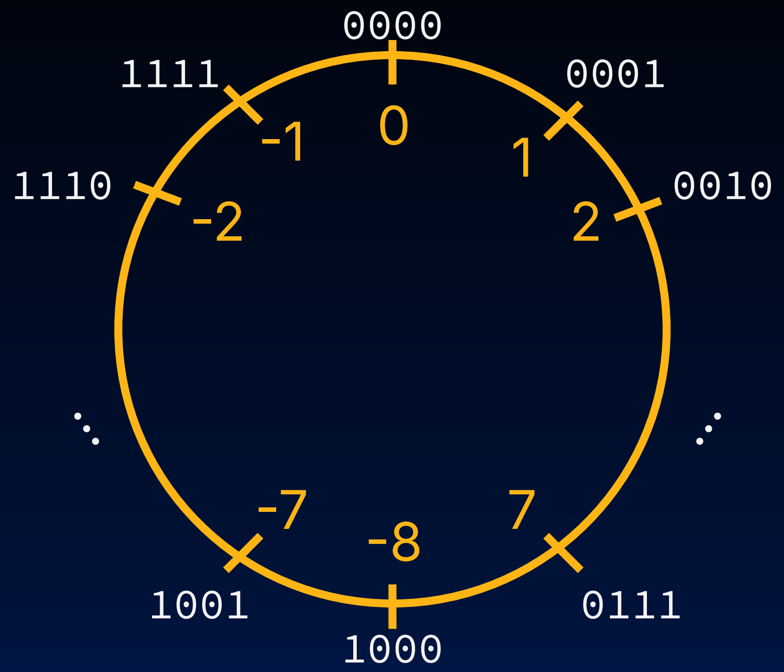
In notes: Read the singular mathematical formula for both positive and negative

On your own: Prove math formula is equivalent to this slide's algorithm

CC BY NC SA

Two's Complement: Addition

Decimal	5
	+ −5
	0
	111 carry bits
Two's complement	0101
	+ 1011
	dropped 10000



Integer overflow in two's complement is a **feature**!
It means we can use the same HW to add numbers of different signs.

Yan, SP26
BY NC SA

Agenda

Bias Encoding

[at home] Used for certain applications

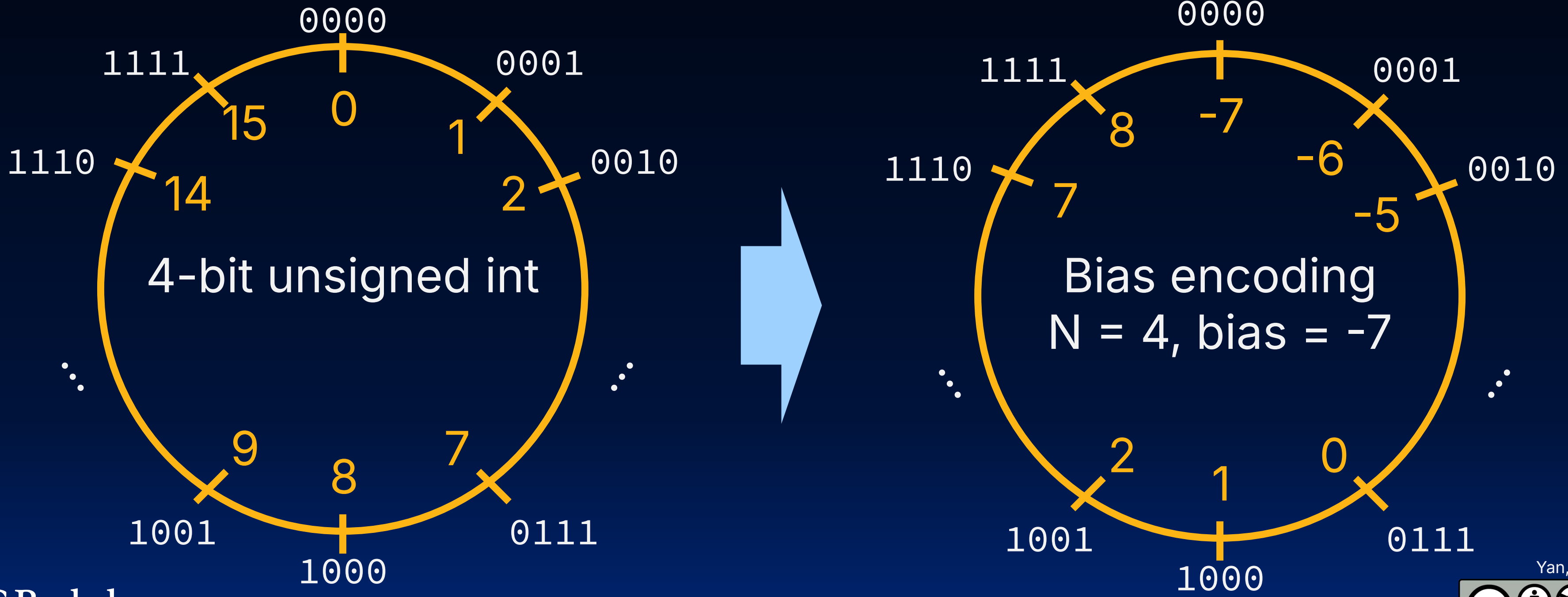
- Binary, Decimal, Hex
- Integer Representations
- Sign-Magnitude, Ones' Complement
- Two's Complement
- Bias Encoding

Yan, SP26

CC BY NC SA

Bias Encoding

- **Bias encoding**: "Shift" the numbers to represent a target range of integers

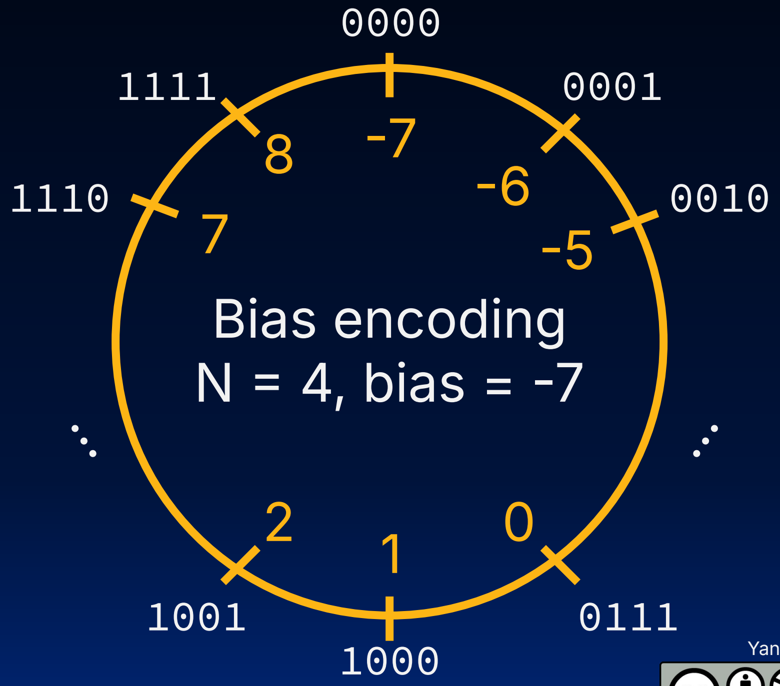


Yan, SP26
CC BY NC SA

Bias Encoding

- **Bias encoding**: “Shift” the numbers to represent a target range of integers

- Number = (unsigned rep) + (bias)
  - 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



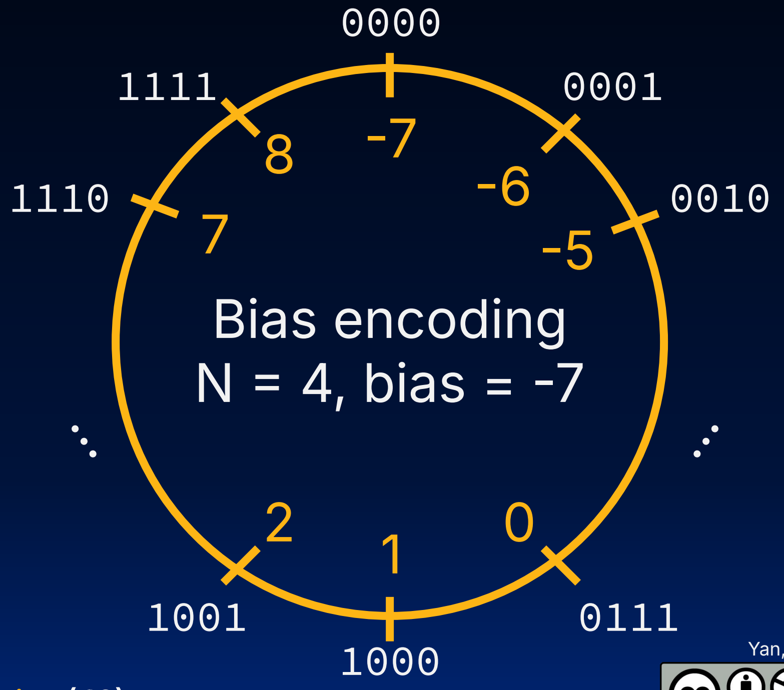
Yan, SP26
CC BY NC SA

When is Bias Encoding Used?

- Number = (unsigned rep) + (bias)
- Useful for, say, AC signals with a DC offset
  - e.g., for 8V and 10V; just store 2V range

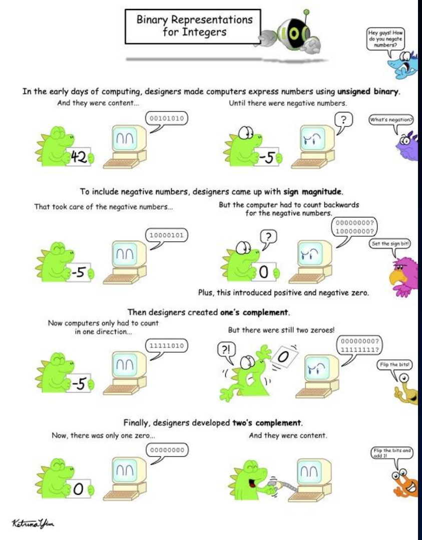
Bias could be anything we want!

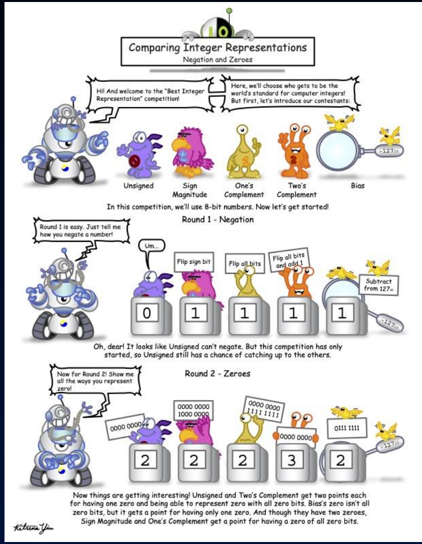
- In practice, with N bits, reasonable choice is bias = $-(2^{N-1} - 1)$
  - E.g., 4 bits bias = $-(2^3-1) = -(8-1) = -7$
- Used for floating point (more later)

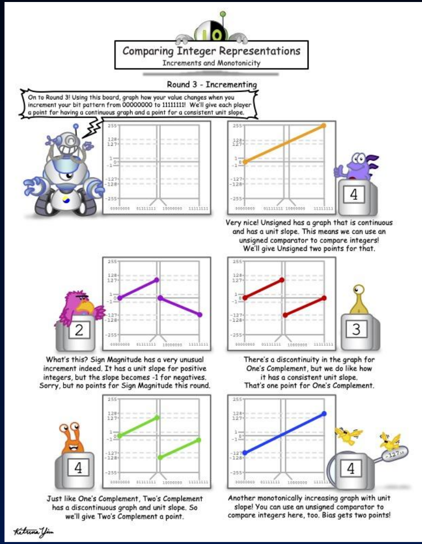


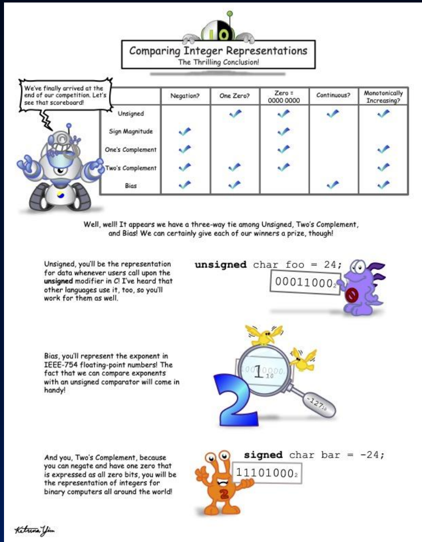
Yan, SP26
cc BY NC SA

Amazing Illustrations by Ketrina (Yim) Thompson









See course notes for reference links!

UC Berkeley

Yan, SP26

BY NC SA

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?

Yan, SP26
CC BY NC SA