

Integers I

CSE 351 Summer 2021

Instructor:

Mara Kirdani-Ryan

Teaching Assistants:

Kashish Aggarwal

Nick Durand

Colton Jobs

Tim Mandzyuk



<http://xkcd.com/257/>

Gentle and Loving Reminders!

- No lecture on Monday 7/5 (campus holiday)
- hw3 due Wednesday 6/30 – 8pm
- Hw4, hw5, lab1a due Friday 7/2 – 8pm
 - I wanted y'all to have a holiday without deadlines!
 - 1 late day on lab1a would take you to Monday, 8pm
- For the rest of the course:
 - Pre-lecture readings are due at 10am
- Other reminders? Does this feel ok?
 - You all belong in computing!

A note on the first survey

- I didn't change everything from last quarter..
 - Booooooooooooooooooooo
- If there's anything that you'd like us to know, difficulties with remote instruction, personal stuff that's coming up, please, please let us know!
 - marakr@cs.washington.edu, if you'd prefer privacy
- This pandemic's been a lot!
 - Last August, I wasn't sure if I could keep living
 - Everyone I know's been through it.
 - Let us know how we can help!

Classroom norms on language?

Reach out with concerns!

**How are y'all feeling
today?**

Seattle's *hot!*

**Submit lab1a by Monday@8pm, only
using one late day!**

Memory, Data, and Addressing

- Representing information as bits and bytes
 - Binary, hexadecimal, fixed-widths
- Organizing and addressing data in memory
 - Memory is a byte-addressable array
 - Machine “word” size = address size = register size
 - Endianness – ordering bytes in memory
- Manipulating data in memory using C
 - Assignment
 - Pointers, pointer arithmetic, and arrays
- **Boolean algebra and bit-level manipulations**

Design Tradeoffs

- When problems don't have a clear answer, you have tradeoffs!
 - Playing card representations
 - Integers (later floats)
- Tradeoffs encode priorities!
 - As well as ideology
- Lots of ideology in 351, emoji to signal ideology?
- When you see ideology (especially when it's unexamined), react with _____



Breakouts!

**Does computing feel like a
safe space?**

**Does it feel oppressive?
...but first...**

Discussion Norms

- Everyone's experience is valid!
- Feelings are valid too!
- Everyone should have space to share!
 - Though, no one's required to share. A “yes” or “no” without explanation is more than enough.
 - Saying “I don't feel like sharing” is good too!
- **I've got a whole heap of hurt here!**
 - Some of y'all might as well
 - Try to be compassionate to everyone!
- A brief grounding...

Breakouts!

- What are your experiences bringing other interests into computing?
- Does computing feel like a safe space?
- Have you felt encouraged to pursue non-CS interests?

Learning Objectives

- Understanding this material means you can...
 - Use bitwise and logical operations to perform bit-level manipulations, $\&$, $|$, \wedge , \sim , \gg , and \ll (lab1a)
 - Convert between decimal and binary representations for signed and unsigned integers
 - Explain tradeoffs between sign-magnitude and two's complement encodings
 - Begin to recognize CS ideology, especially when considering tradeoffs between different designs
 - Develop a sense of how knowledge insulation affect CS, and more importantly, how knowledge insulation affects you!

Boolean Algebra

- Developed by George Boole in 19th Century
 - Algebraic representation of logic (T \rightarrow 1, F \rightarrow 0)
 - AND: $A \& B = 1$ when both A is 1 and B is 1
 - OR: $A | B = 1$ when either A is 1 or B is 1
 - XOR: $A \wedge B = 1$ when either A is 1 or B is 1; **not both**
 - NOT: $\sim A = 1$ when A is 0 and vice versa
 - DeMorgan's law: $\sim (A | B) = \sim A \& \sim B$

AND		
$\&$	0	1
0	0	0
1	0	1

OR		
$ $	0	1
0	0	1
1	1	1

XOR		
\wedge	0	1
0	0	1
1	1	0

NOT	
\sim	
0	1
1	0

General Boolean Algebras

❖ Operate on bit vectors

- Operations applied bitwise
- All of the properties of Boolean algebra apply

$$\begin{array}{rclcl}
 01101001 & 01101001 & 01101001 & & \\
 \& 01010101 & | 01010101 & ^ 01010101 & \sim 01010101 \\
 \hline
 01000001 & 01111101 & 00111101 & & 10101010
 \end{array}$$

❖ Examples of useful operations:

$$x \wedge x = 0$$

$$\begin{array}{r}
 01010101 \\
 ^ 01010101 \\
 \hline
 00000000
 \end{array}$$

$$x | 1 = 1, \quad x | 0 = x$$

$$\begin{array}{r}
 01010101 \\
 | 11110000 \\
 \hline
 11110101
 \end{array}$$

Bit-Level Operations in C

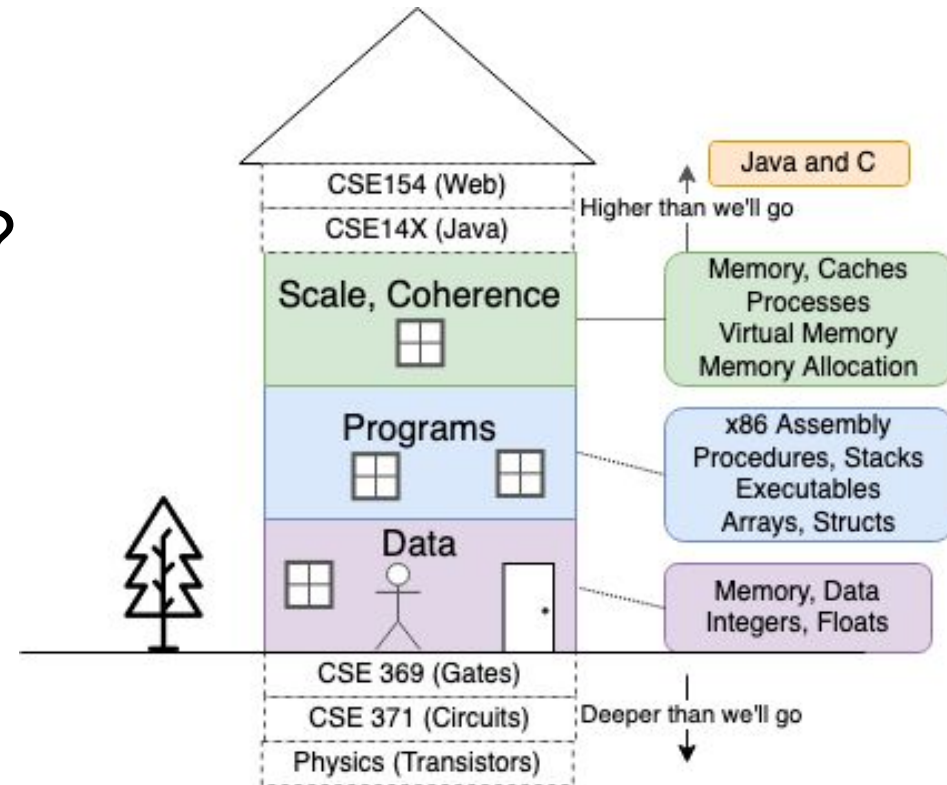
- $\&$ (AND), $|$ (OR), \wedge (XOR), \sim (NOT)
 - View arguments as bit vectors, apply operations bitwise
 - Apply to any “integral” data type
 - long, int, short, char, unsigned
- Examples with `char a, b, c;`
 - `a = (char) 0x41; // 0x41->0b 0100 0001`
`b = ~a; // 0b 1011 1110 ->0xBE`
 - `a = (char) 0x69; // 0x69->0b 0110 1001`
`b = (char) 0x55; // 0x55->0b 0101 0101`
`c = a & b; // 0b 0100 0001 ->0x41`
 - `a = (char) 0x41; // 0x41->0b 0100 0001`
`b = a; // 0b 0100 0001`
`c = a ^ b; // 0b 0000 0000 ->0x0`

Contrast: Logic Operations

- Logical ops in C: `&&` (AND), `||` (OR), `!` (NOT)
 - 0 is False, anything nonzero is True
 - Always return 0 or 1
 - **Early termination** (short-circuit evaluation) of `&&`, `||`
- Examples (`char` data type)
 - `!0x41 -> 0x00`
 - `!0x00 -> 0x01`
 - `!!0x41 -> 0x01`
 - `p && *p`
 - If `p` is the **null pointer** (`0x0`), then `p` is never dereferenced!
 - `0xCC && 0x33 -> 0x01`
 - `0x00 || 0x33 -> 0x01`

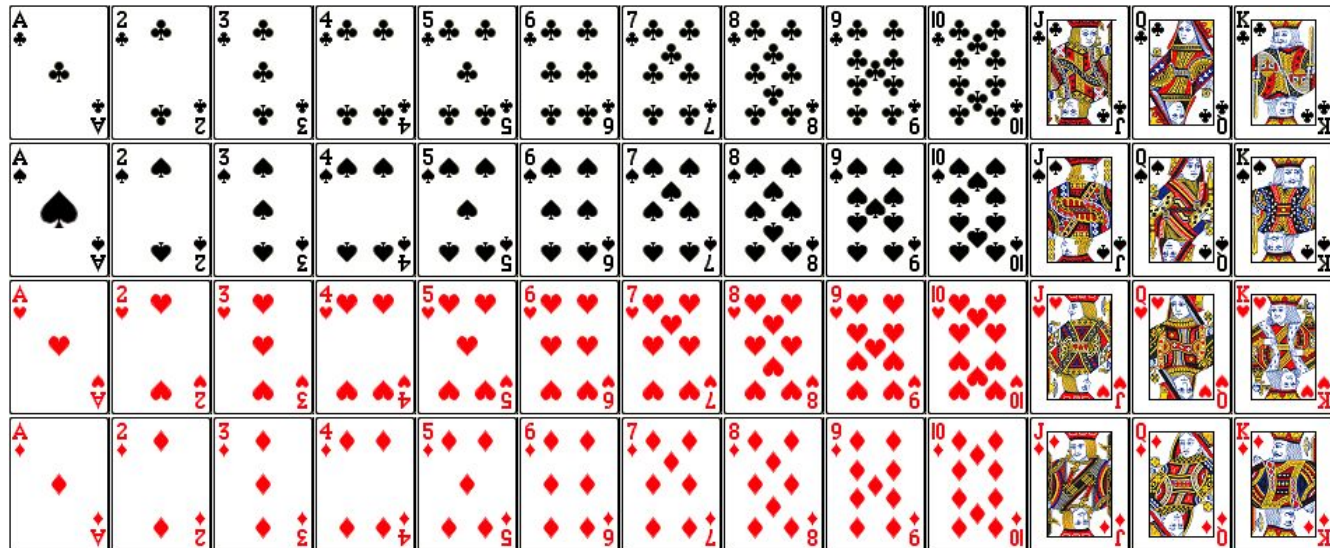
First Floor: Data

- How do we represent data (strings, numbers) computationally?
- What limits exist? Why?
- What values were encoded into data representations?
- What was prioritized? Why?
- Today: Integers!



But before we get to integers....

- Encode a standard deck of playing cards
 - 52 cards in 4 suits
 - How do we encode suits, face cards?
- What operations do we want to make easy to implement?
 - Which is the higher value card?
 - Are they the same suit?



Two possible representations

1. 1 bit/card (52): bit that refers to card set to 1



low-order 52 bits of 64-bit word

- “One-hot” encoding (similar to set notation)
- Drawbacks:
 - Hard to compare values and suits
 - Large number of bits required



4 suits 13 numbers

2. 1 bit/suit (4), 1 bit/number (13): 2 bits set

- Pair of one-hot encoded values
- Easier to compare suits/values; but lots of bits used

Two better representations

3) Binary encoding of all 52 cards – 6 bits needed

- $2^6 = 64 \geq 52$
- Fits in one byte (smaller than one-hot encodings)
- How can we make value and suit comparisons easier?



low-order 6 bits of a byte





4) Separate encodings of suit (2 bits), value (4 bits)



suit value

- Also fits in one byte; easy to do comparisons

K	Q	J	...	3	2	A
1101	1100	1011	...	0011	0010	0001

	00
	01
	10
	11

Design Tradeoffs

- We went from 52 bits to 6 bits
 - Are 6 bit representations “better” than 52 bit ones?
 - It depends!!! What are we prioritizing?
- In general, folks prioritize along their *ideology*
 - CS: Efficiency, minimalism
 - Frequently choice between space/speed, without considering socio-technical factors
- There usually isn't a “perfect” decision!



Compare Card Suits

```
char hand[5];           // represents a 5-card hand
char card1, card2;      // two cards to compare
card1 = hand[0];
card2 = hand[1];
...
if ( sameSuitP(card1, card2) ) { ... }
```

```
#define SUIT_MASK  0x30
```

```
int sameSuitP(char card1, char card2) {
    return (!(card1 & SUIT_MASK) ^ (card2 & SUIT_MASK));
    //return (card1 & SUIT_MASK) == (card2 & SUIT_MASK);
}
```

SUIT_MASK = 0x30 =

0	0	1	1	0	0	0	0
---	---	---	---	---	---	---	---

suit value

Compare Card Sui

mask: a bit vector designed to achieve a desired behavior when used with a bitwise operator on another bit vector v .

Here we turn all *but* the bits of interest in v to 0.

```
char hand[5];           // represents a 5-card hand
char card1, card2;      // two cards to compare
card1 = hand[0];
card2 = hand[1];
...
if ( sameSuitP(card1, card2) ) { ... }
```

```
#define SUIT_MASK 0x30 // 0011 0000
```

```
int sameSuitP(char card1, char card2) {
    return (!((card1 & SUIT_MASK) ^ (card2 & SUIT_MASK)));
    //return (card1 & SUIT_MASK) == (card2 & SUIT_MASK);
}
```

returns **int**

SUIT_MASK = 0x30 =

0	0	1	1	0	0	0	0
---	---	---	---	---	---	---	---

equivalent

suit value

Compare Card Suits

mask: a bit vector designed to achieve a desired behavior when used with a bitwise operator on another bit vector v .

Here we turn all *but* the bits of interest in v to 0.

```
#define SUIT_MASK 0x30
```

```
int sameSuitP(char card1, char card2) {
    return (!((card1 & SUIT_MASK) ^ (card2 & SUIT_MASK)));
    //return (card1 & SUIT_MASK) == (card2 & SUIT_MASK);
}
```

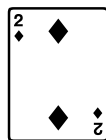
0 0 0 1 0 0 1 0

&

0 0 1 1 0 0 0 0

=

0 0 0 1 0 0 0 0



SUIT_MASK

0 0 0 1 1 1 0 1

&

0 0 1 1 0 0 0 0

=

0 0 0 1 0 0 0 0

^

0 0 0 0 0 0 0 0

!

0 0 0 0 0 0 0 1

! (x ^ y) equivalent to x == y

Compare Card Values

mask: a bit vector designed to achieve a desired behavior when used with a bitwise operator on another bit vector v .

```
char hand[5];           // represents a 5-card hand
char card1, card2;      // two cards to compare
card1 = hand[0];
card2 = hand[1];
...
if ( greaterValue(card1, card2) ) { ... }
```

```
#define VALUE_MASK  0x0F // 0000 1111

int greaterValue(char card1, char card2) {
    return ((unsigned int)(card1 & VALUE_MASK) >
            (unsigned int)(card2 & VALUE_MASK));
}
```

VALUE_MASK = 0x0F =

0	0	0	0	1	1	1	1
---	---	---	---	---	---	---	---

suit

value

Compare Card Values

mask: a bit vector designed to achieve a desired behavior when used with a bitwise operator on another bit vector v .

```
#define VALUE_MASK 0x0F

int greaterValue(char card1, char card2) {
    return ((unsigned int)(card1 & VALUE_MASK) >
            (unsigned int)(card2 & VALUE_MASK));
}
```

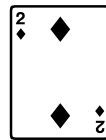
0	0	1	0	0	0	1	0
---	---	---	---	---	---	---	---

&

0	0	0	0	1	1	1	1
---	---	---	---	---	---	---	---

=

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---



VALUE_MASK

0	0	1	0	1	1	0	1
---	---	---	---	---	---	---	---

&

0	0	0	0	1	1	1	1
---	---	---	---	---	---	---	---

=

0	0	0	0	1	1	0	1
---	---	---	---	---	---	---	---

$2_{10} > 13_{10}$

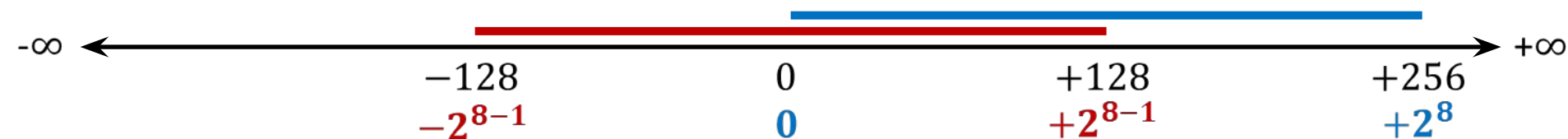
0 (false)

Integers

- **Binary representation of integers**
 - **Unsigned and signed**
- Shifting and arithmetic operations – for Lab 1a
- In C: Signed, Unsigned and Casting
- Consequences of finite width representations
 - Overflow, sign extension
- CS sprung out of mathematics
 - Creators wanted to match existing axioms

Encoding Integers

- ❖ The hardware (and C) supports two flavors of integers
 - *unsigned* – only the non-negatives
 - *signed* – both negatives and non-negatives
- ❖ Cannot represent all integers with w bits
 - Only 2^w distinct bit patterns
 - Unsigned values: $0 \dots 2^w - 1$
 - Signed values: $-2^{w-1} \dots 2^{w-1} - 1$
- ❖ **Example:** 8-bit integers (*e.g.* `char`)



Unsigned Integers; “standard” binary

- Unsigned values follow the standard base 2 system
 - $b_7b_6b_5b_4b_3b_2b_1b_0 = b_72^7 + b_62^6 + \dots + b_12^1 + b_02^0$
- Add and subtract using the normal “carry” and “borrow” rules, just in binary

63	00111111
+ 8	+ 00001000
<hr/> 71	<hr/> 01000111

- Useful : $2^{N-1} + 2^{N-2} + \dots + 2 + 1 = 2^N - 1$
 - *i.e.* N ones in a row = $2^N - 1$

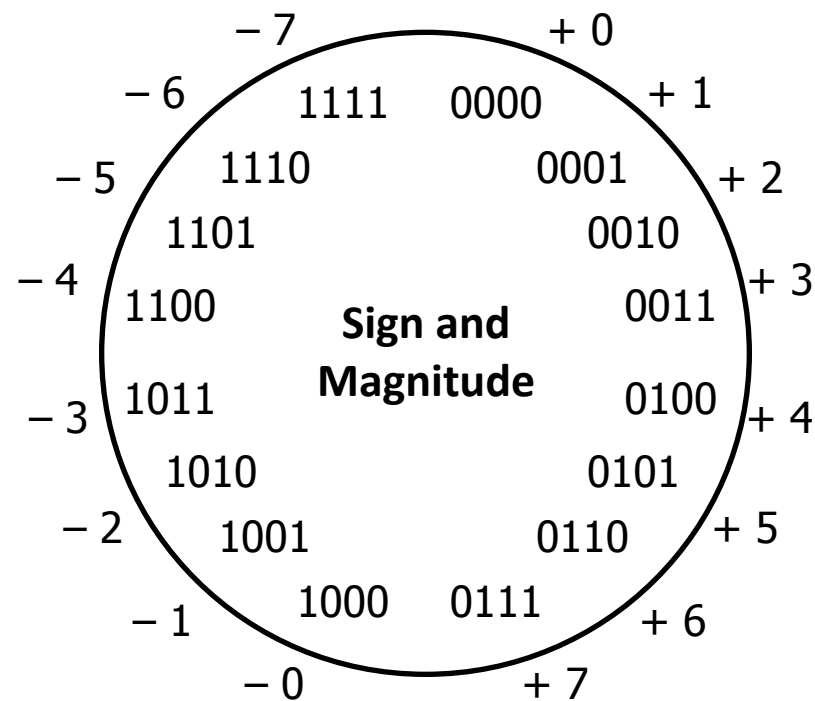
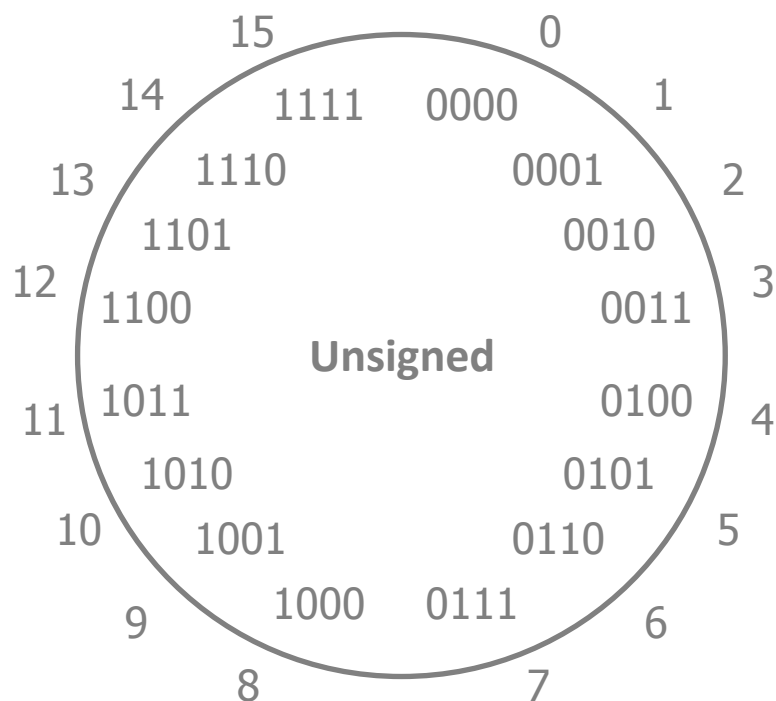
Sign-Magnitude Encoding

Most Significant Bit

- Designate the high-order bit (MSB) as “sign bit”
 - $\text{sign}=0$: positive #'s; $\text{sign}=1$: negative #'s
- Benefits:
 - Using MSB as sign bit matches positive numbers with unsigned
 - All zeros encoding is still = 0
- Examples (8 bits):
 - $0x00 = 00000000_2$ is non-negative; the sign bit is 0
 - $0x7F = 01111111_2$ is non-negative ($+127_{10}$)
 - $0x85 = 10000101_2$ is negative (-5_{10})
 - $0x80 = 10000000_2$ is negative... .. zero???

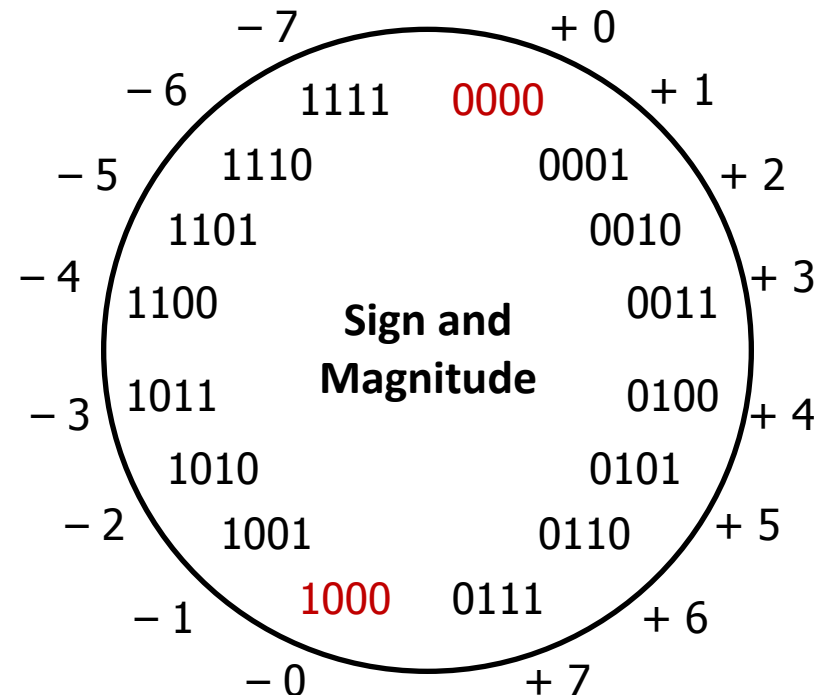
Sign-Magnitude Encoding

- MSB is the sign bit, rest of the bits are magnitude
- Drawbacks?



Sign-Magnitude Encoding

- MSB is the sign bit, rest of the bits are magnitude
- Drawbacks:
 - **Two representations of 0** (bad for checking equality)



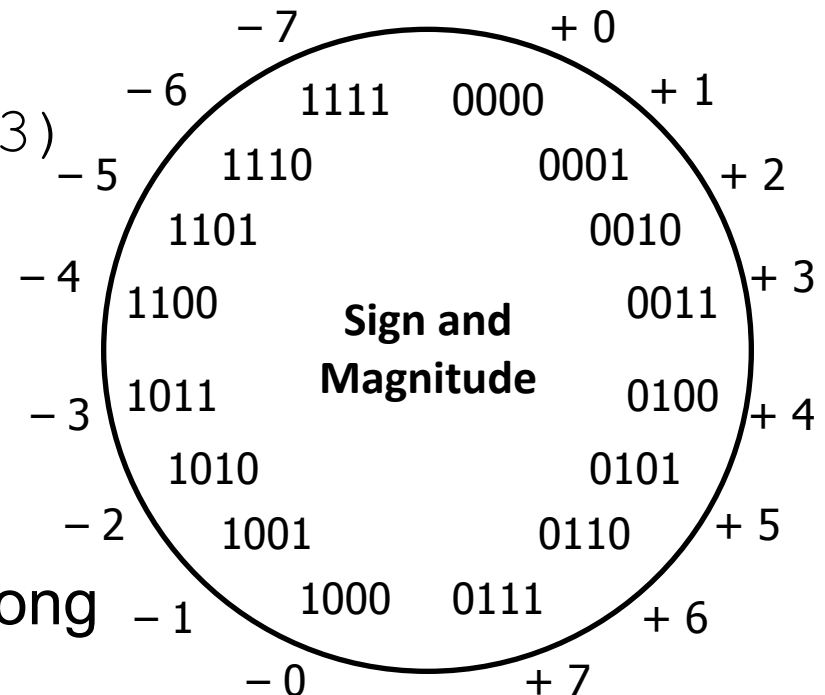
Sign-Magnitude Encoding

- MSB is the sign bit, rest of the bits are magnitude
- Drawbacks:
 - Two representations of 0 (bad for checking equality)
 - Arithmetic is cumbersome**
 - Example: $4 - 3 \neq 4 + (-3)$

4	0100
<u>- 3</u>	<u>- 0011</u>
1	0001



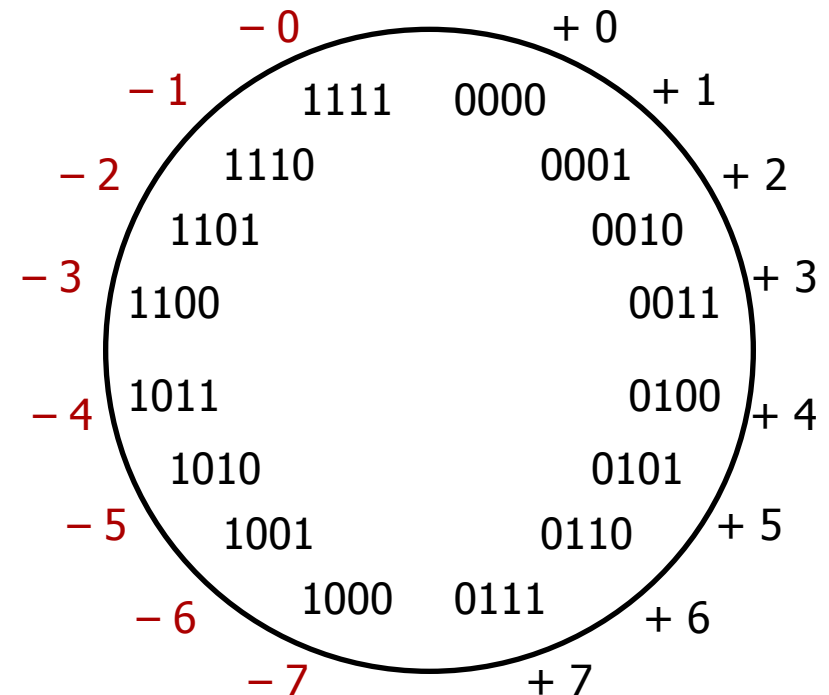
4	0100
<u>+ -3</u>	<u>+ 1011</u>
-7	1111



- Negatives “increment” in wrong direction!

Two's Complement

- Let's fix these problems:
 - “Flip” negative encodings so incrementing works



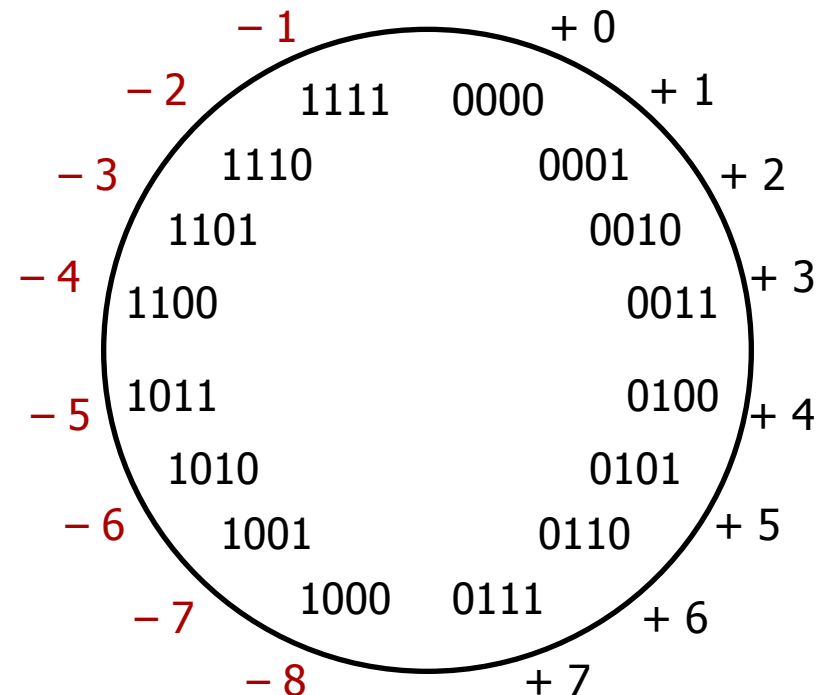
Two's Complement

❖ Let's fix these problems:

- 1) "Flip" negative encodings so incrementing works
- 2) "Shift" negative numbers to eliminate -0

❖ MSB *still* indicates sign!

- This is why we represent one more negative than positive number (-2^{N-1} to $2^{N-1}-1$)



Two's Complement Negatives

- Accomplished with *one neat* (mathematical) *trick!*

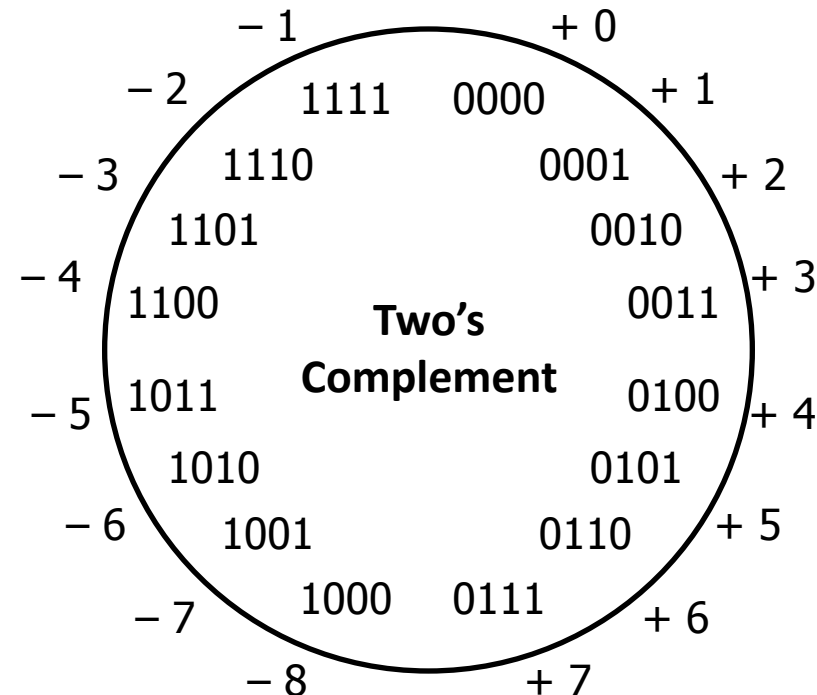
b_{w-1} has weight -2^{w-1} , other bits have usual weights $+2^i$



- 4-bit Examples:

- 1010_2 unsigned:
 $1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 10$

- 1010_2 two's complement:
 $-1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = -6$



Two's Complement Negatives

- Accomplished with *one neat* (mathematical) *trick!*

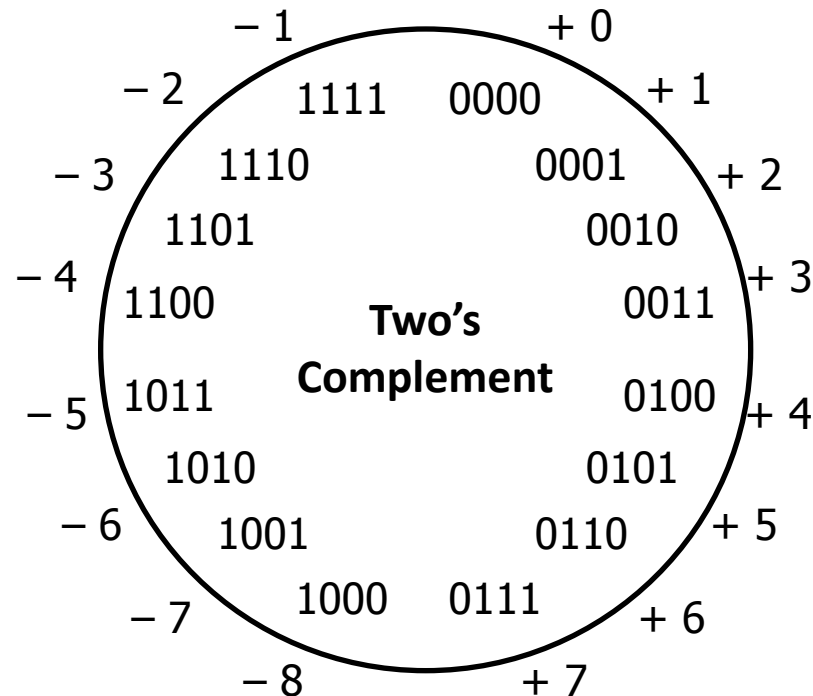
b_{w-1} has weight -2^{w-1} , other bits have usual weights $+2^i$



- 1 represented as:

$$1111_2 = -2^3 + (2^3 - 1)$$

- MSB makes it super negative!
- Add up all the other bits to get back up to -1



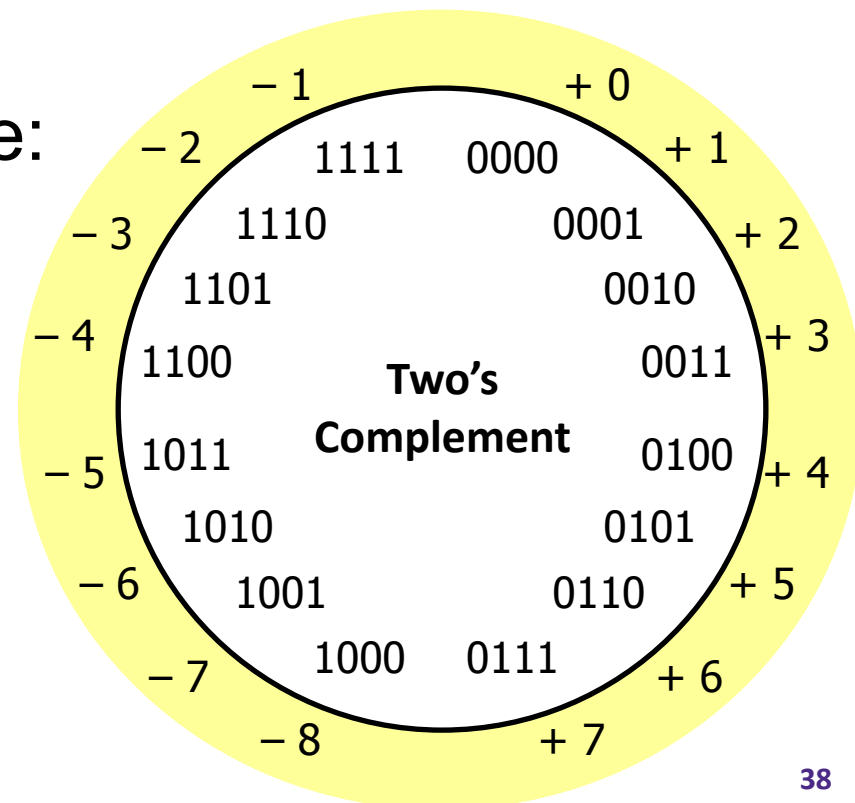
Why Two's Complement is So Great

- Roughly same number of (+) and (−) numbers
- Positive number encodings match unsigned
- Single zero encoding, all zeros

- Simple negation procedure:

- Get negative representation of any integer by taking bitwise complement and then adding one!

$$(\sim x + 1 == -x)$$



Polling Question [Int I - b]

- Take the **4-bit number encoding** $x = 0b1011$
- What value would this encoding have in...
 - Unsigned? Sign and Magnitude? Two's Complement?



-4



-5



11



-3



Help!

Integers

- Binary representation of integers
 - Unsigned and signed
- **Shifting and arithmetic operations** – (Lab 1a)
- In C: Signed, Unsigned and Casting
- Consequences of finite width representations
 - Overflow, sign extension

Shift Operations

- Left shift ($x \ll n$) bit vector x by n positions
 - Throw away (drop) extra bits on left
 - Fill with 0s on right
- Right shift ($x \gg n$) bit-vector x by n positions
 - Throw away (drop) extra bits on right
 - Logical shift (for **unsigned** values)
 - Fill with 0s on left
 - Arithmetic shift (for **signed** values)
 - Replicate most significant bit on left
 - Maintains sign of x

Shift Operations

- Left Shift ($x \ll n$)
 - Fill with 0s on right
- Right shift ($x \gg n$)
 - Logical shift (for **unsigned** values)
 - Fill with 0s on left
 - Arithmetic shift (for **signed** values)
 - Replace with MSB on left
 -
- Notes
 - Shift by $n < 0$ or $n \geq w$ are *undefined*
 - In C: behavior of \gg is determined by the compiler
 - Depends on type of x (signed/unsigned)

	x	0010 0010
	$x \ll 3$	0001 0 000
logical:	$x \gg 2$	00 00 1000
arithmetic:	$x \gg 2$	00 00 1000

	x	1010 0010
	$x \ll 3$	0001 0 000
logical:	$x \gg 2$	00 10 1000
arithmetic:	$x \gg 2$	11 10 1000

Shifting Arithmetic?

- What are the following computing?
 - $x \gg n$
 - $0b\ 0100 \gg 1 = 0b\ 0010$
 - $0b\ 0100 \gg 2 = 0b\ 0001$
 - Divide by 2^n
 - $x \ll n$
 - $0b\ 0001 \ll 1 = 0b\ 0010$
 - $0b\ 0001 \ll 2 = 0b\ 0100$
 - Multiply by 2^n
- Shifting “faster” than multiply/divide operations

Left Shifting Arithmetic 8-bit Example

- No difference in left shift operation for unsigned and signed numbers (just manipulates bits)
 - Difference comes during interpretation: $x * 2^n$?

		Signed	Unsigned
$x = 25;$	00011001 =	25	25
$L1 = x \ll 2;$	0001100100 =	100	100
$L2 = x \ll 3;$	00011001000 =	-56	200
$L3 = x \ll 4;$	000110010000 =	-112	144

signed overflow

unsigned overflow

Right Shifting Arithmetic 8-bit Examples

- **Reminder:** C operator `>>` does *logical* shift on **unsigned** values and *arithmetic* shift on **signed** values

- **Logical** Shift: $x / 2^n$?

`xu = 240u;` 11110000 = 240

`R1u=xu>>3;` 00011110000 = 30

`R2u=xu>>5;` 0000011110000 = 7

rounding
(down)

Right Shifting Arithmetic 8-bit Examples

- **Reminder:** C operator `>>` does *logical* shift on **unsigned** values and *arithmetic* shift on **signed** values

- **Arithmetic** Shift: $x / 2^n$?

`xs = -16;` 11110000 = -16

`R1s=xs>>3;` 11111110000 = -2

`R2s=xs>>5;` 111111110000 = -1

rounding
(down)

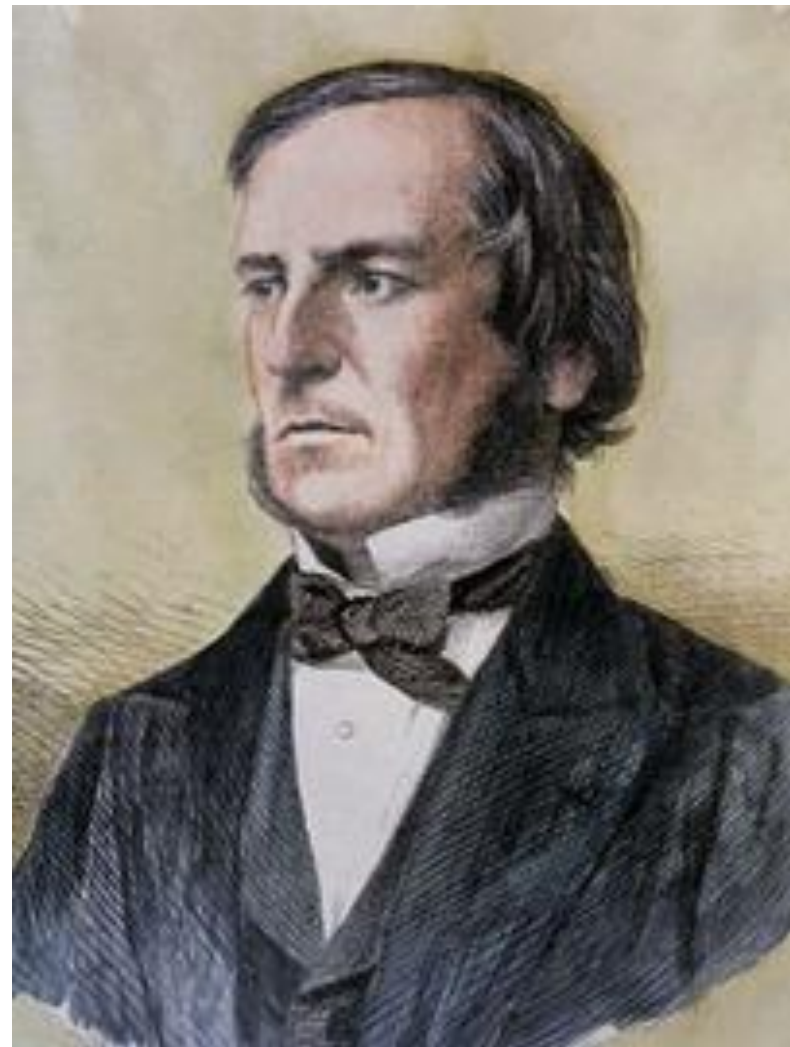
Summary

- Bit-level operators allow for fine-grained manipulations of data
 - Bitwise AND (`&`), OR (`|`), and NOT (`~`) different than logical AND (`&&`), OR (`||`), and NOT (`!`)
 - Especially useful with bit masks
- Choice of *encoding scheme* is important
 - Tradeoffs based on size requirements and desired operations
- Integers represented using unsigned and two's complement representations
 - Limited by fixed bit width
 - We'll examine arithmetic operations next lecture
- Shifting is a useful bitwise operator
 - Right shifting can be arithmetic (sign) or logical (0)
 - Can be used in multiplication with constant or bit masking

Knowledge & Insulation

George Boole

- A committed educator!
 - Walked in the rain to teach, taught in wet clothes, wife wrapped him in wet blankets to cure him
- Philosophy, logic
 - Pseudo-religious doctrine of psychology
 - HEAVILY influenced by Indian systems of logic



Colonialism in 19th Century

- 18th /19th century India under colonial rule
 - Re-education camps, the whole business

“On examining this work I saw in it, not merely merit worthy of encouragement, but merit of a peculiar kind, the encouragement of which, as it appeared to me, was likely to promote native effort towards the restoration of the native mind in India.”

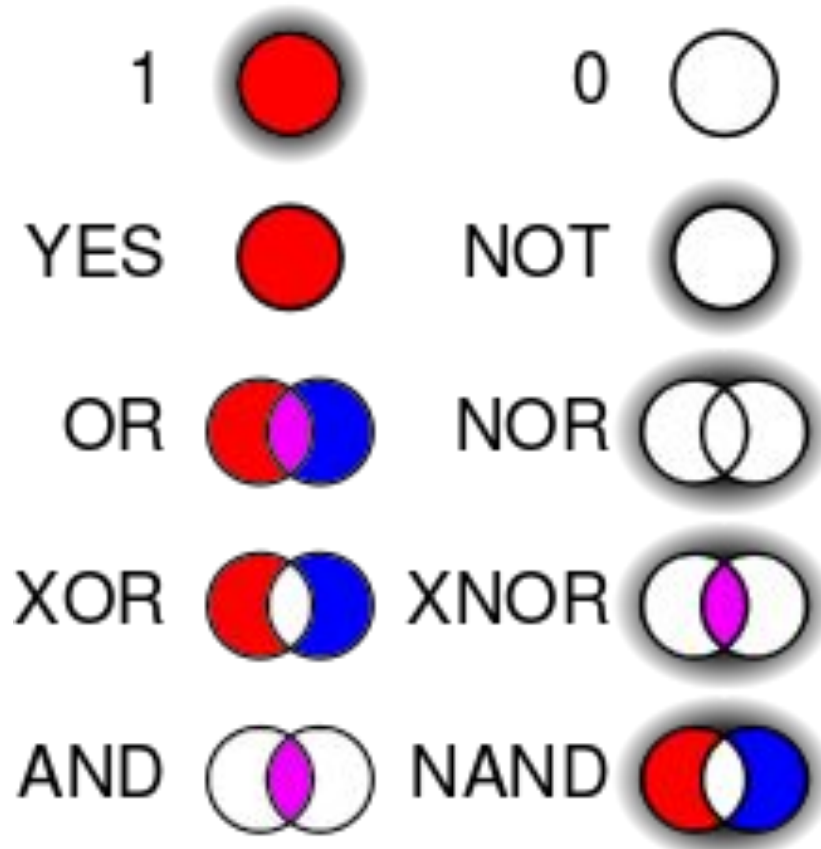
DeMorgan, prefacing: “A treatise on problems of maxima and minima, solved by algebra.” by Ram Chundra, European Publication in 1859

Claude Shannon (1916-2001)

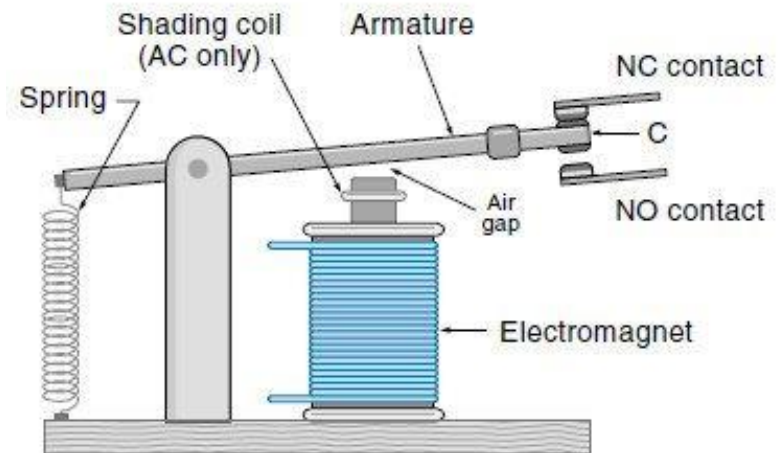
- Mathematician
 - Took a philosophy class! Not CS!
 - Learned Boole's algebra, realized relevance to digital logic
- 1937 Master's Thesis
 - "Founded" information theory



I mean, it's a close connection



An electromagnetic relay.



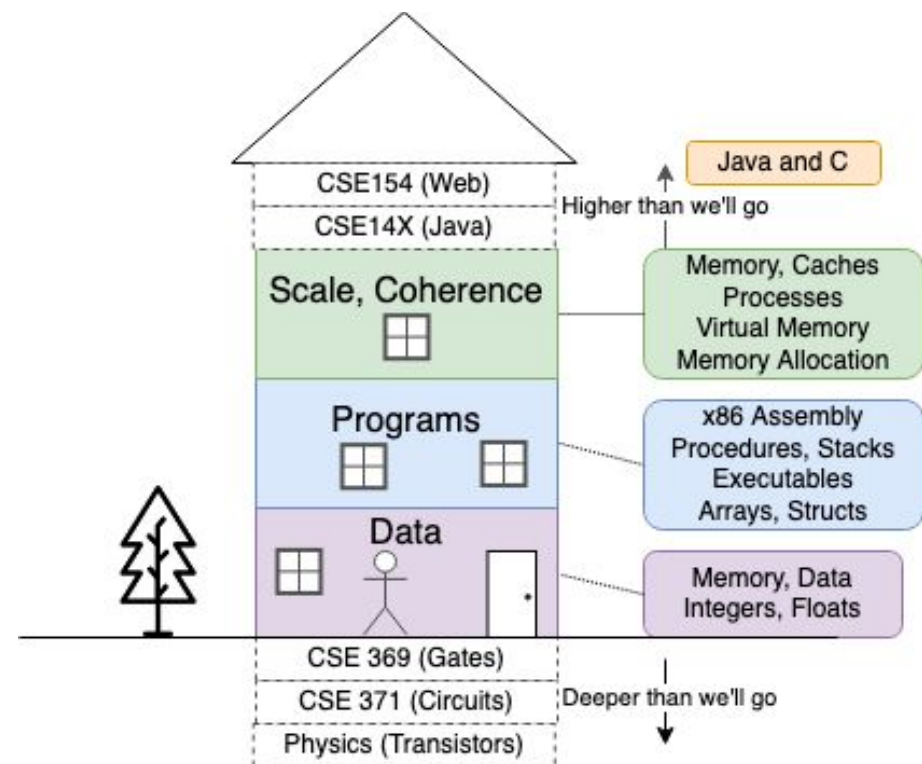
(a) Parts of the relay

*Ba dum **dum***

The “best” ideas come from outside CS!

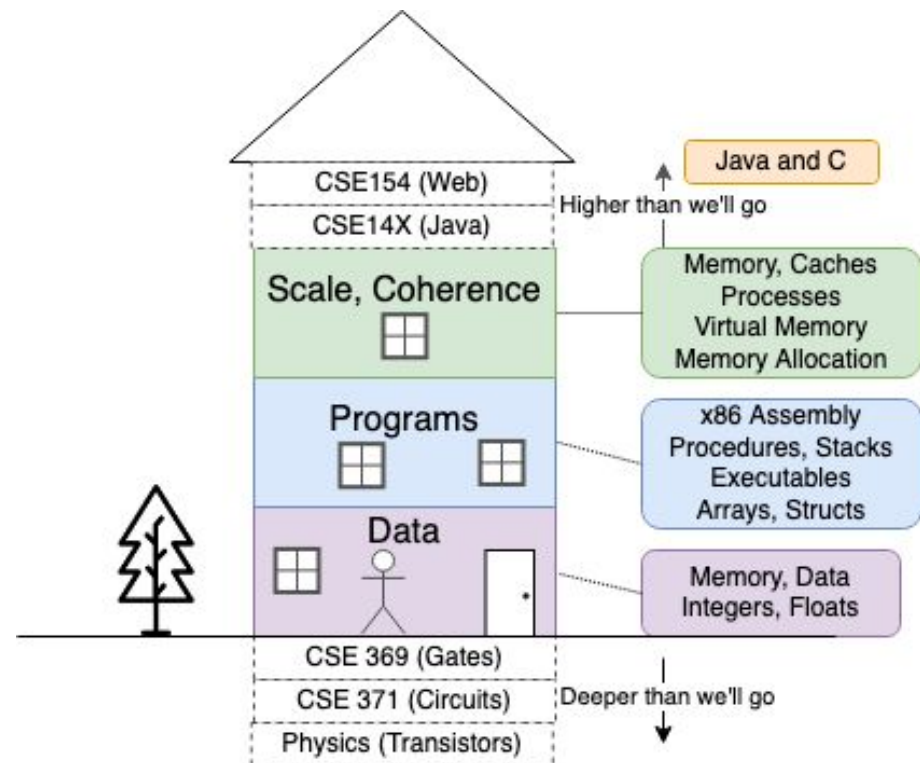


Insulation: a metaphor



Insulation: a metaphor

- It keeps us warm!
 - It protects us from the outside!
- It can be harmful!
 - Asbestos, clearly
 - Fiberglass is still bad
 - Modern alternatives still harmful



**Insulation keeps us
*comfortable and
protected.***

**I'd argue that CS is
insulated!
But, why?**

Amy's Keynote

- Briefly, CS has been a safe space for:
 - Women, whose jobs were automated by computers
 - Bullied kids that liked calculators
 - Wealthy “young geeks” (me)
 - Closeted trans kids (me)
 - **Autistic people (me)**
 - Queer folks trying to move to more progressive cities (SF/Seattle)
 - Those seeking social mobility

Amy's Keynote

- Briefly, CS has been a safe space for:
 - Women, whose jobs were automated by computers
 - Bullied kids that liked calculators
 - Wealthy “young geeks” (me)
 - Closeted trans kids (me)
 - **Autistic people (me)**
 - Queer folks trying to move to more progressive cities (SF/Seattle)
 - Those seeking social mobility
- **Y'all aren't the target audience, what'd you think?**

Autism, extremely briefly



Invisible Strings

@M_Kelter

In the last few decades, we have detected thousands of planets outside of our solar system. They did not suddenly appear at the time we found them. No one calls it an "exoplanet epidemic". The planets were always there, just previously undiscovered. This is a tweet about autism.

More information @ <https://autisticadvocacy.org/about-asan/about-autism/>

Autism, extremely briefly

- Special interests: intense, passionate fixations on topics, “little professors”
 - Turing was “obsessed” with codes and ciphers, didn’t care about much else
- Theory of mind (some low, some high):
 - Understanding that the brains of others are different than yours
- Empathy (some low, some high):
 - I have to be careful who I’m holding space for, others not so much
- Communication
 - Neurodiverse folks tend to communicate well with each other, less well with neurotypical folks

Autism, and computing

- Attention to detail highly valued in CS
- Historically, CS is an intellectual space centering rules (theory) and routines (algorithms)
- Little social interaction required, “don’t bother me” assumed
- Lots of space to pursue special interests without interruption!
- **Most of the world expect neurotypical conformity, most of CS emphasizes conformity along some* aspects of neurodiversity**
 - Again, UW is much better than my undergrad

Insulation and me

- I identify as neurodiverse (autistic), trans, enby
- CS ~~is~~ was a great closet...
 - Encouraged to be "in my head", outside my body
 - Focus on patterns, abstract concepts
 - Completely ignore society
- ...but, really, that's no way to live
 - I promise, it would've killed me
- At some point, leave the house and go exploring!
 - Though camping can still be a stretch 😊

I promise, I'm not alone

I promise, I'm not alone

- Generally, transgender rate of 0.5% to 1%
- Autism Spectrum Disorder (ASD) around 1.5%
- 8-15% ASD among trans* folks!
- Higher rates of ASD in STEM than elsewhere
 - Most CS folks tend not to recognize neurodiversity
- I got to undergrad and realized that lots of folks were like me, realized later that most of them were autistic.

Making Space in CS

- Foundations of computing:
 - Insulating for safety (i.e. being trans in Seattle, at UW)
 - Exploring new frontiers (even just philosophy)
- Both foundational!
 - Though, many focus on one or the other
 - Technical/Socio-technical; you need both!
- CS needs both to survive and thrive!
 - New ideas and safe spaces both necessary

Making Space in CS



Exploring Space, Making Space

- Goodness, if you can, take classes outside CS
 - Even better if they're non-technical!
 - *Philosophy, Art, History, **Education??***
- Claude Shannon, *Founder of Information Theory*
 - Go take a class because it looks interesting!
 - Maybe it'll be relevant to computing?
 - More likely, you'll learn something about yourself!

Insulation is Necessary and Deadly!

Try to make space when you can!