Data Representation

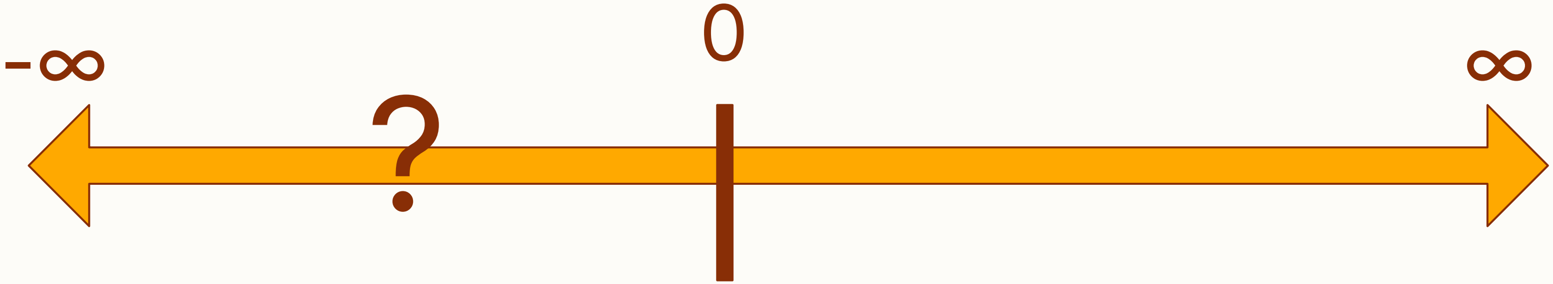
Reese Hatfield

Vocabulary

- Bit = 1/0
- Byte
  - Refers to 8 contiguous bits
- Nibble
  - 4 Contiguous bits
- 4 bytes → 32 bits

Data Representation

- We can represent 0→∞ in binary
  - Practically, we only usually do 2^64
- But there are more numbers
- How about negatives



Signed Magnitude

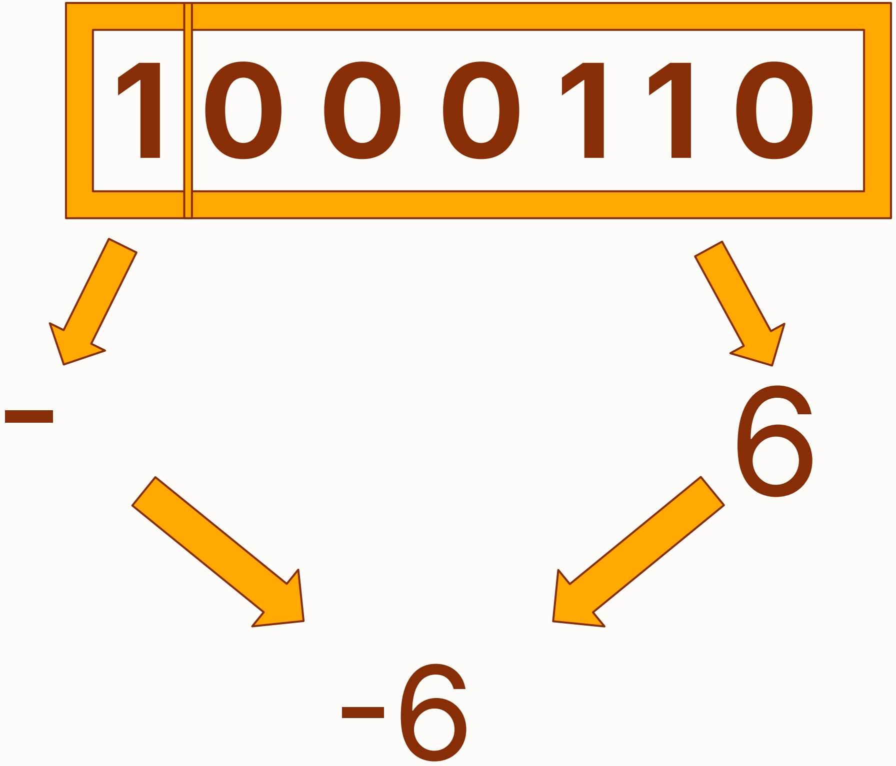
- You have a few options
- Simplest approach
- Have a bit represent the sign
- If MSB == 1
  - Negative
- If MSB == 0
  - Positive

Signed Magnitude

- Signed magnitude representation
  - MSB = Sign
  - Everything else = Magnitude
    - Distance from 0

$\boxed{1\,0\,0\,0\,0\,1\,1}$

Signed Magnitude



Signed Magnitude

- This has a few problems
  - Different bit lengths = different numbers
  - What is this number?

$\boxed{1000}$ = -0 ?

Signed Magnitude

- Signed Magnitude addition
  - Completely broken

	1010	- 2
+	0010	+ 2
	1100	= -4 ?

2’s Complement

- Is there a better option?
- Yes
  - 2’s complement binary
- Similar representation
- MSB = Sign Bit
- Remaining = Count down instead

2's Complement

- Same bit length problem as before
  - 4-bit representation
  - Different from 8-bit
- Consider the range of this system
- -8 → +7

1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1

0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7

Signed Magnitude

- 2's Complement Addition
  - Works as intended
- Lets see that in action
  - Use previous example

- Addition works like normal
- Discard any carry bits

	1	1			
	0	0	1	0	+2
+	1	1	1	0	-2
1	0	0	0	0	= 0

Discard any carry bits

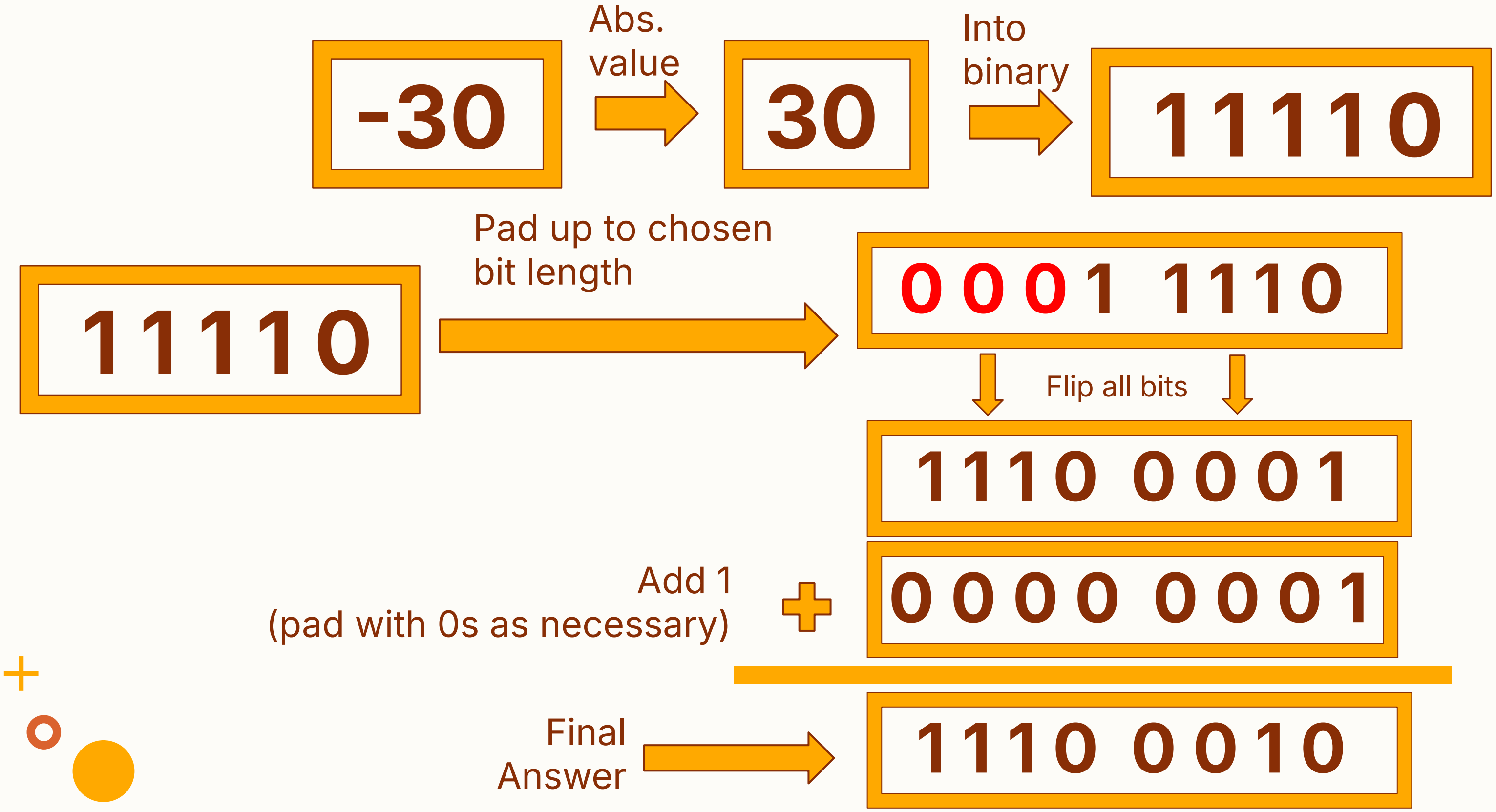
2’s Complement

- How to go from Decimal to 2′s Complement Binary
  - **Chose a bit length first**
  - If number is positive
    - Just convert like normal
  - What to do if negative?

2's Complement

- Negative Decimal to 2's complement
  - **Chose a bit length**
  - Take abs. value of number
  - Convert to binary like normal
  - Flip all the bits
  - Add 1_2

2’s Complement



2's Complement

- Example together
  - -26_{10}
- Whiteboard example
  - -23_{10} 8-bit two's complement
  - Hint: $23_{10} = 10111_2$

2’s Complement

- What about the other direction?
- 2’s complement bin. ⇒ Decimal
- Let’s reframe how we think of the MSB
  - It is effectively a sign bit
- However
  - It really represents the negative version of the next exponents
  - Let’s see that

- MSB is really the “number of -8s”
- Or whatever the MSB is

-8	4	2	1
1	**1**	**0**	**1**

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

2’s Complement

- You can use this same process to convert
- Let’s do an example
- Whiteboard: 1011 0111

Negation

- How do we go turn a positive 2's complement number into a negative?
- Simple
  - Flip bits
  - Add 1

1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1

0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0

Negation

- We have already done this when converting

- What are we really doing there?
  - Flip the sign
  - Account for off by 1
- Please Remember:
  - Different bit lengths = Different output

Other Number Representations

Reese Hatfield

Other Representations

- Data Representation
  - Binary = a number
- Could we do more?
  - What other things could we represent?
- More numbers
- Colors
- Characters + more

Real numbers

- We can represent all integers
- What about real numbers
  - Decimals, etc
- Limited number of bits
- Infinite amount of numbers
- Compromises

Fixed Point

- We could just allocate some bits to the decimal
- Continue having 2^N
- Use negative exponents

2^2	2^1	2^0		2^-1	2^-2	2^-3
1	0	1	.	0	0	1

Fixed Point

2^2	2^1	2^0		2^-1	2^-2	2^-3
1	0	1	.	1	0	1
2^2		2^0		2^-1		2^-3
4	+	1	+	0.5	+	0.125

5.625

Fixed Point

- Decimal Point ⇛ Binary Point
- Consider the range of a 6-bit systems
  - Only integers
  - 0-64
- What's our new range?
  - 0-8
  - With some added precision

Fixed Point

- When we lock out “binary point” into a single position
- We lose a significant amount of range
- How do we fix this?
  - Make out point “floating” instead of being fixed in place

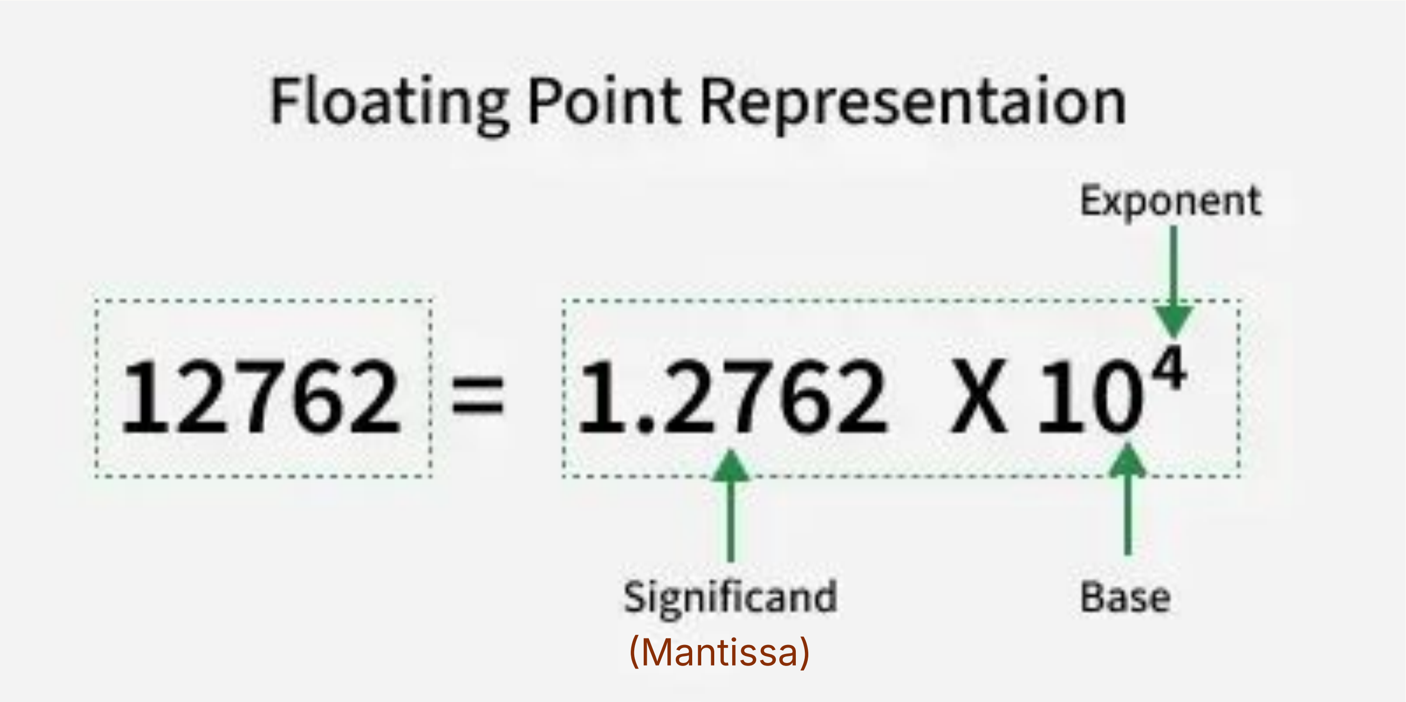
Floating Point

- Many ways to do this
- IEEE 745
  - Defines the “floating point” standard
  - Every programming language uses this
- Makes a lot of compromises, but fixes range problem
- Let’s see how it works

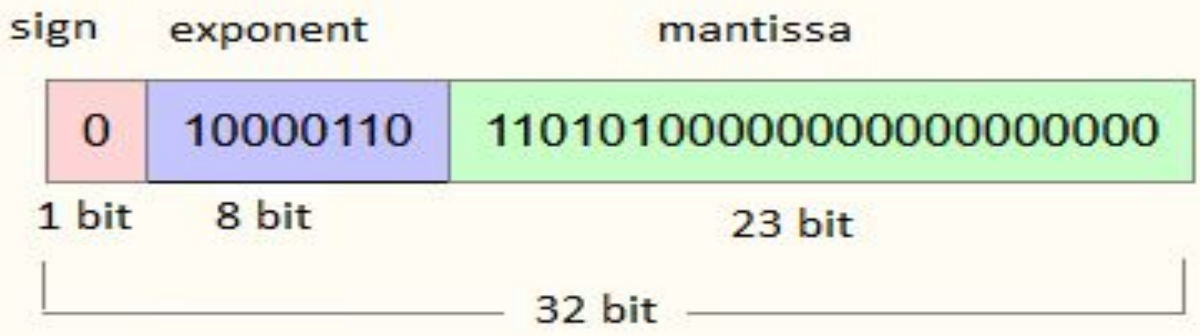
Floating Point

- Scientific notation
    - Provides a quick way of writing very very large numbers
- Uses exponents to shorten things
- Floating points also takes advantage of this

Floating Point



- Sign bit (1-bit)
  - Sign of the number
- Exponent
  - 10 ^ whatever power
- Mantissa
  - Number to multiply by



Floating Point

- A lot more clever than we will go into
- Has some problems
  - Let’s add some number (0.1)x3
- Floating point math errors
- We cannot represent all real numbers
- Compromises
  - Math is sometimes off
  - NaN, -0, 1.0 / +-0.0

Floating Point

- Designing this is really hard
- Kind of the best we got
- Can be expanded
  - Often uses 64 bits instead of 32
- Problems arise from
  - Limited bits
  - Infinite numbers

Other Representations

- Characters
    - How to represent them?
- Could define a mapping
    - $\text{Number}_2 \Rightarrow \text{Character}$
    - ~Logical mapping
- Encoding Standard
    - Any way of turning data into something meaningful

ASCII

- ASCII
  - American Standard Code for Information Interchange
- 1961
  - Up until now, everyone was kind of just using their own “standard”

ASCII

- Simple Standard
  - 7-bit standard
  - 128 possibilities
  - 0-128 in binary
- 65-90 Uppercase Characters
- 97-122 Lowercase Character

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

ASCII

- Lots of legacy characters
- Cool tricks
  - Flip 1-bit to convert from different cases
- Extremely limited
- English is primary language of computing
  - We still need to use other languages

ASCII

- Unicode
  - Fixes this limited problem
  - Supports different languages
  - Universal standard (kinda)
  - Supports emojis!

- “Unicode Miracle”

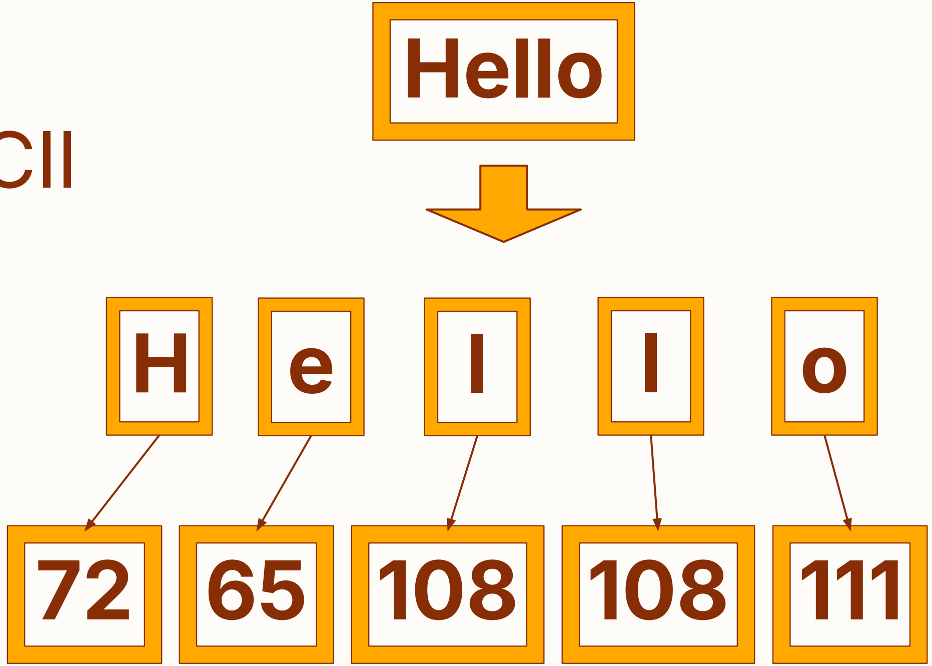
ASCII

- The first 128 unicode characters = ASCII
- ASCII is a subset of Unicode

- Generally, people just consider plaintext ASCII

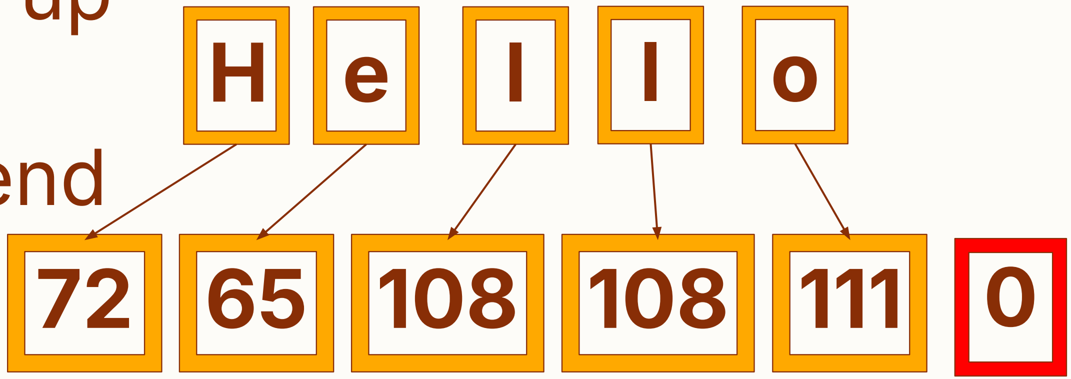
ASCII

- Convert a message to ASCII
- Lookup in table
- This is CASE SENSITIVE



ASCII

- List of characters
- Is called a String
  - This comes up later
- Strings often end with a NULL character



Colors

- We can use bits to represent theoretically anything
  - ~Real numbers
  - Characters
- Colors is also a common one
  - “8-bit color”

Colors

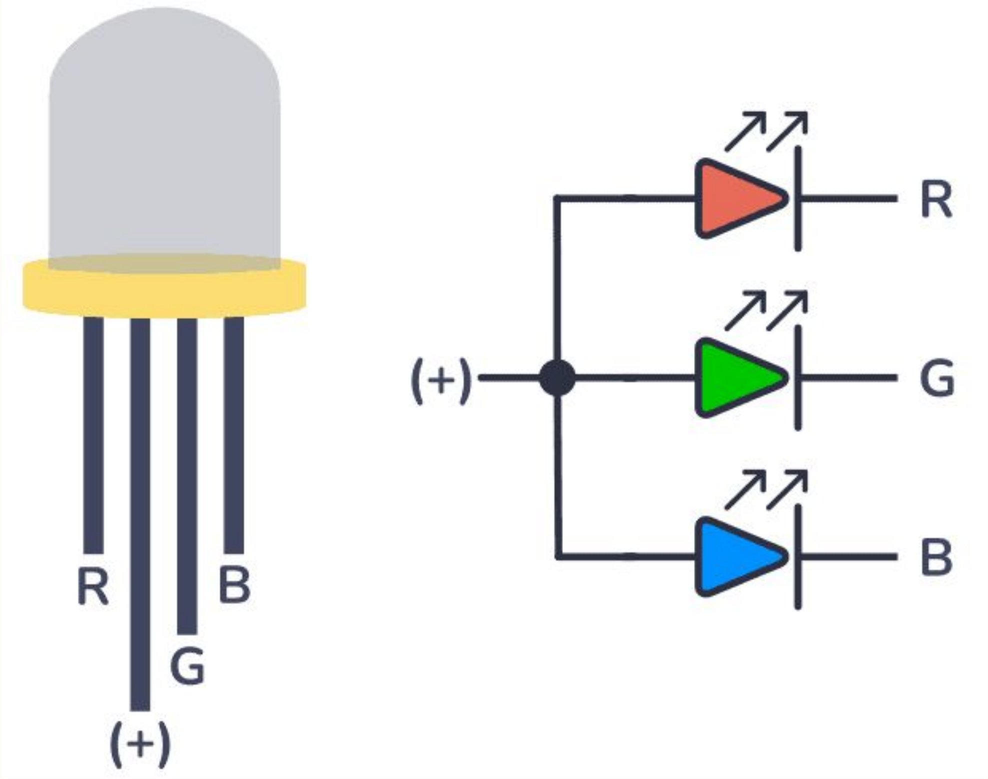
- Color and light is really complicated
  - Bits are very simple
  - Compromises
- Additive colors
- Mix different light colors
  - Make more complex colors

Colors

- Additive Colors
  - Red Channel
  - Green Chanel
  - Blue Channel
- Allocate bits to each
- You could then map these to physical lights

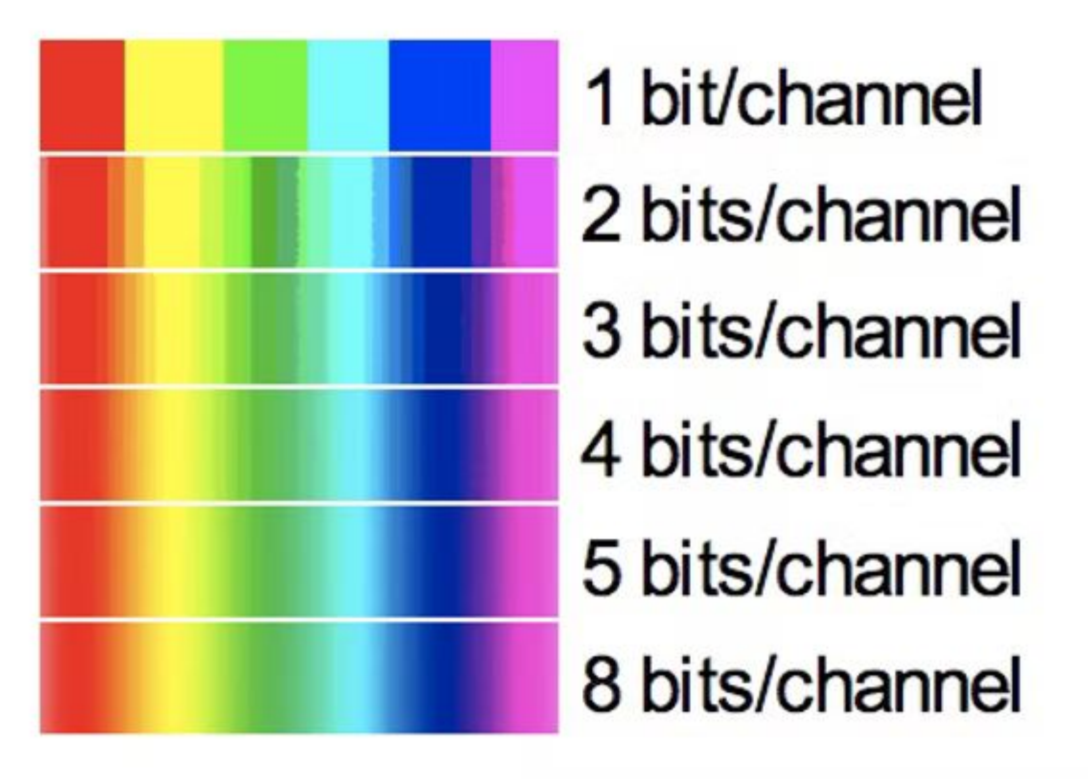
Colors

- Physical hardware
- Mapped to each channel
- Your screen has 1000s of these



Colors

- More bits = more colors
- 8 bits is usually good enough
- 256 values per channel



Colors

- This lets us describe color using only 3 numbers
- Color = (R, G, B)
- E.g. Green = (0, 255, 0)
  Purple = (255, 0, 255)
- Let's use a color picker
- There are different color formats as well (e.g CMYK)

Colors

- We use these colors to encode images
- What about transparency?
- Sometimes a 4th channel is added
- “Alpha” value → RGBA
- Let’s see that

What are we doing?

- Taken binary
  - Easy for computers to work with
- Developed “standards”
- Different ways of **encoding** information with bits

Encodings

- Unsigned binary integers
- Signed magnitude
- Two's Complement
- Fixed Point
- Floating Point
- RGB(A)

Encodings

- All represent different potentially useful ideas with bits
- What are standards?
- Who defines them?
- Who gets to choose which bit goes where?

Standards

- “Standards” are really just documents
- Let’s look at some
- IEEE ⇒ Floating point

- GS1 ⇒ Barcode

Logic (Gates)

Reese Hatfield

Logic Gates

- Everything we've done so far has been "theoretical"
- Bits are how we *could* interact with a computer
- Next, we'll dive into how a computer DOES interact with bits

Logic Gates

- Bell Labs (AT&T)
  - 1947
- Scientists wanted to build a re-usable amplifier
- Vacuum tubes were horrible at the time
- Wanted easy, solid, reusable circuit

Logic Gates

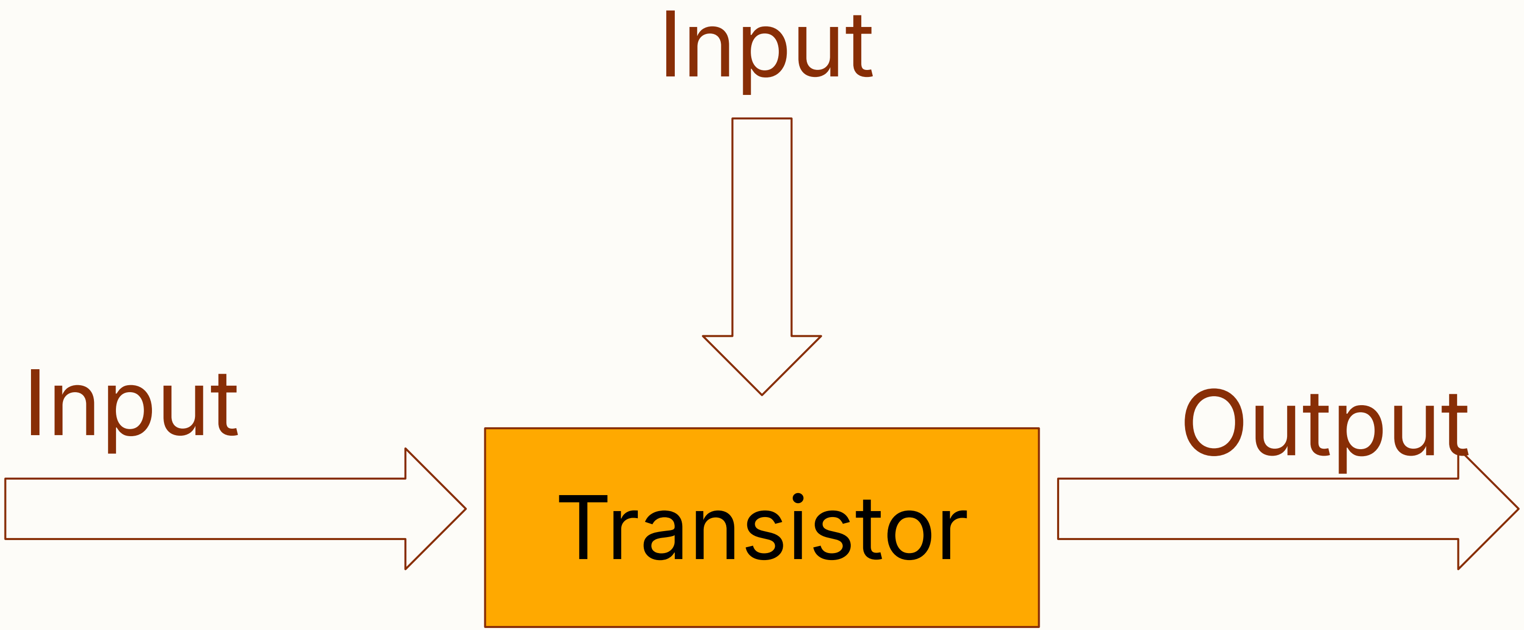
- The Transistor is born
- Bulky at the time
- Size reduced dramatically over time

Logic Gates

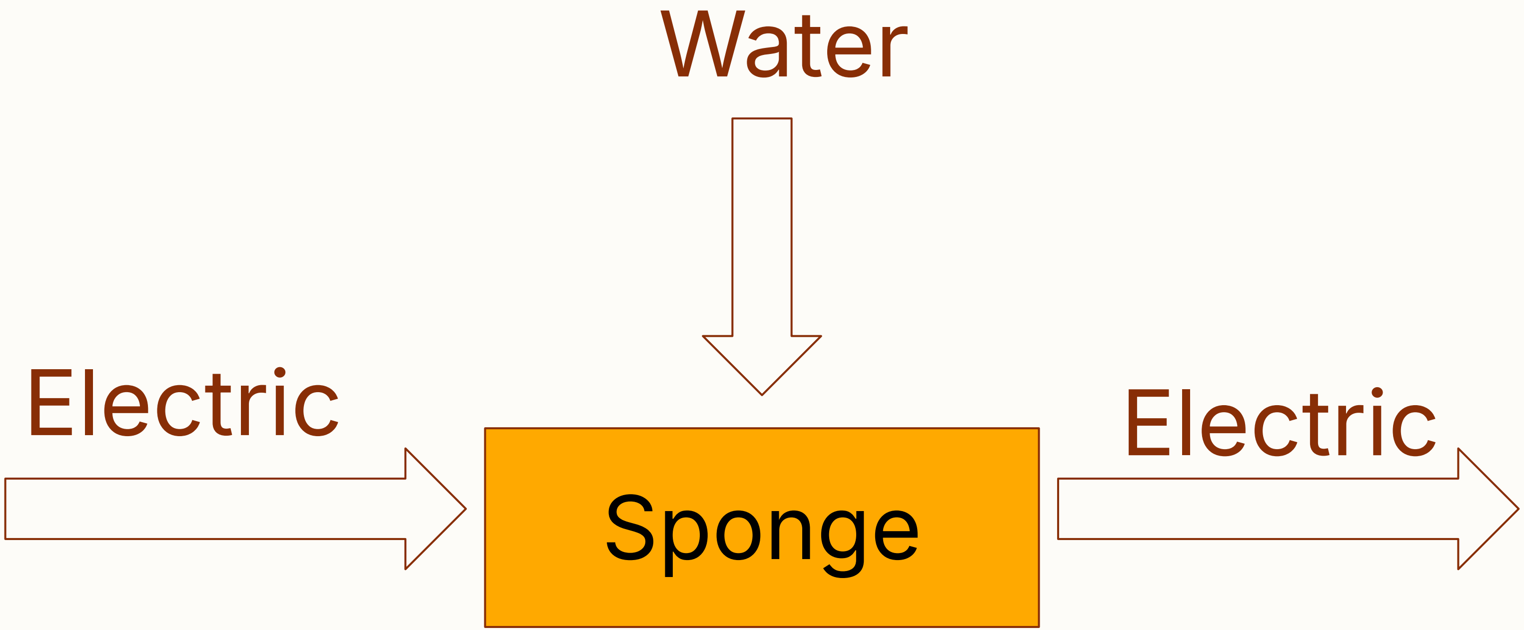
Logic Gates

- So what actually is a transistor?
- Semi-conductor
- An analogy
  - Dry Sponge
  - Wet Sponge
- 2 inputs, 1 output
- Let’s look at that

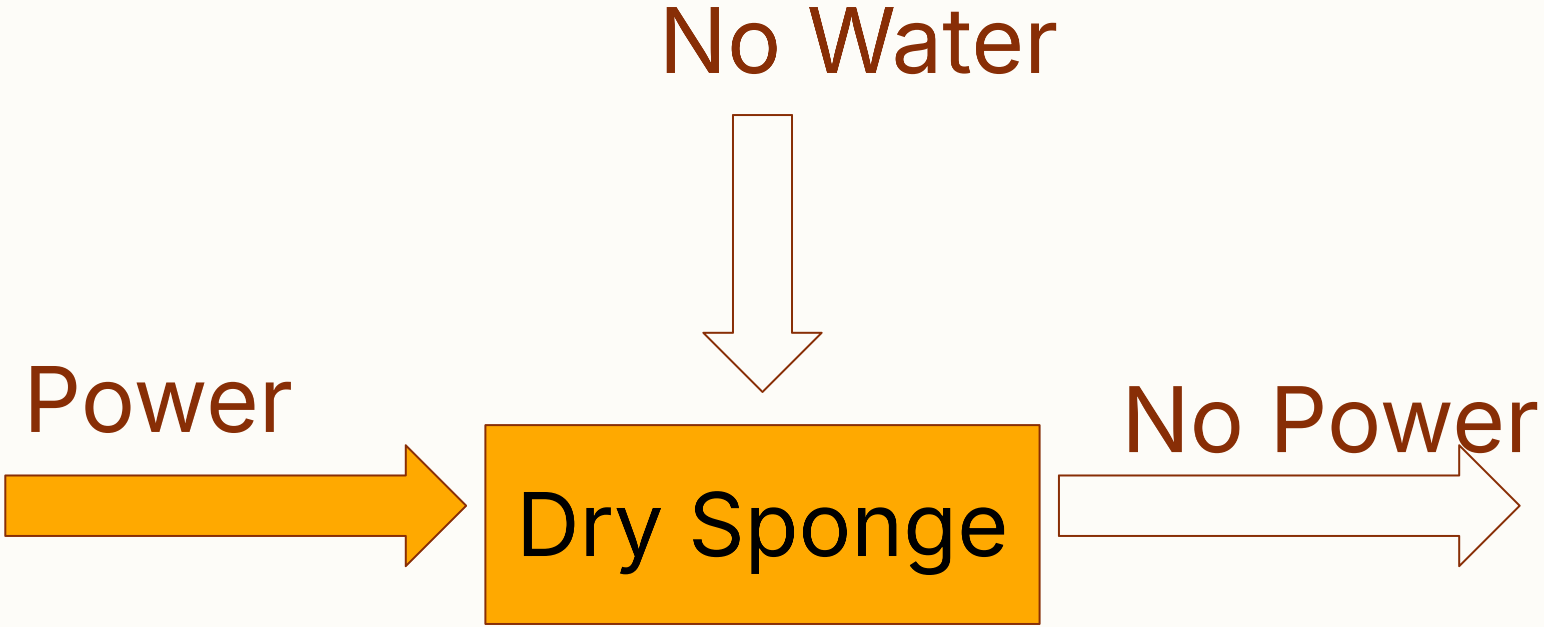
Logic Gates



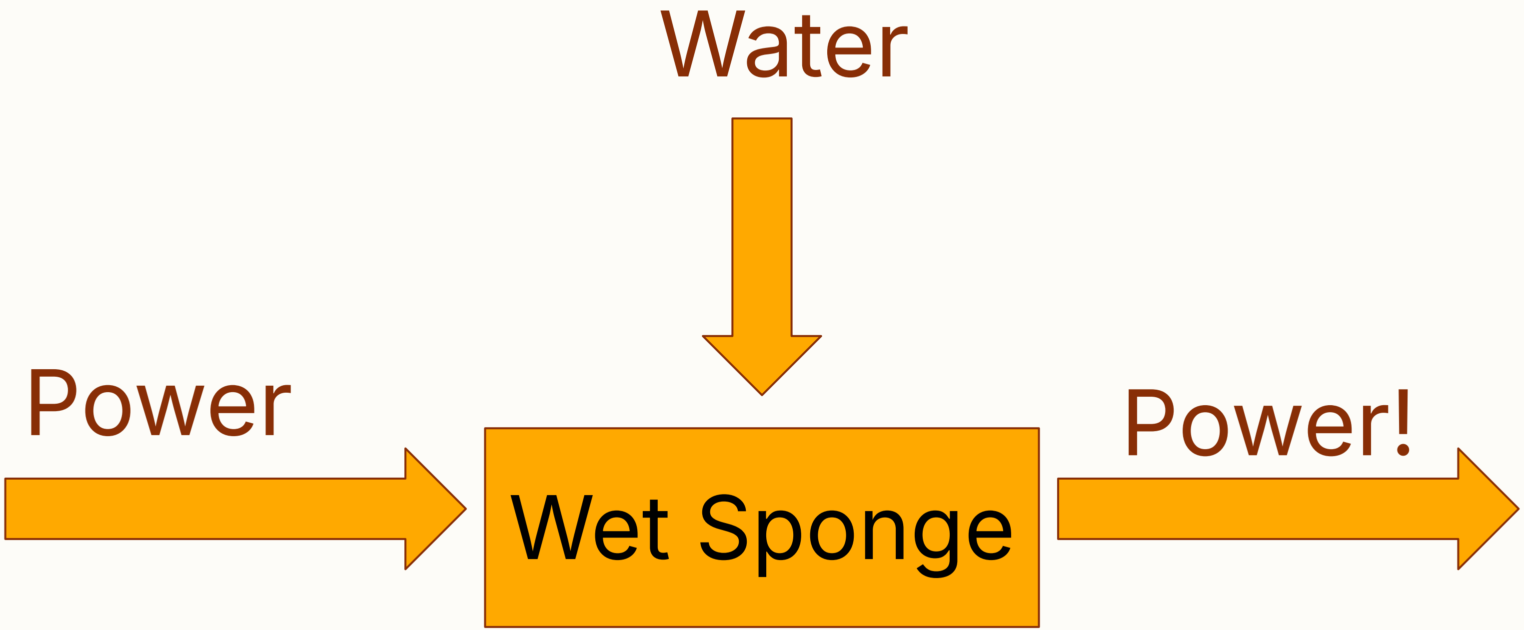
Logic Gates



Logic Gates



Logic Gates



Logic Gates

- Today, these are incredibly tiny
- Power/No Power
- Water/ No Water
- Examples of bits!

- Think back to that RBG device we saw last week