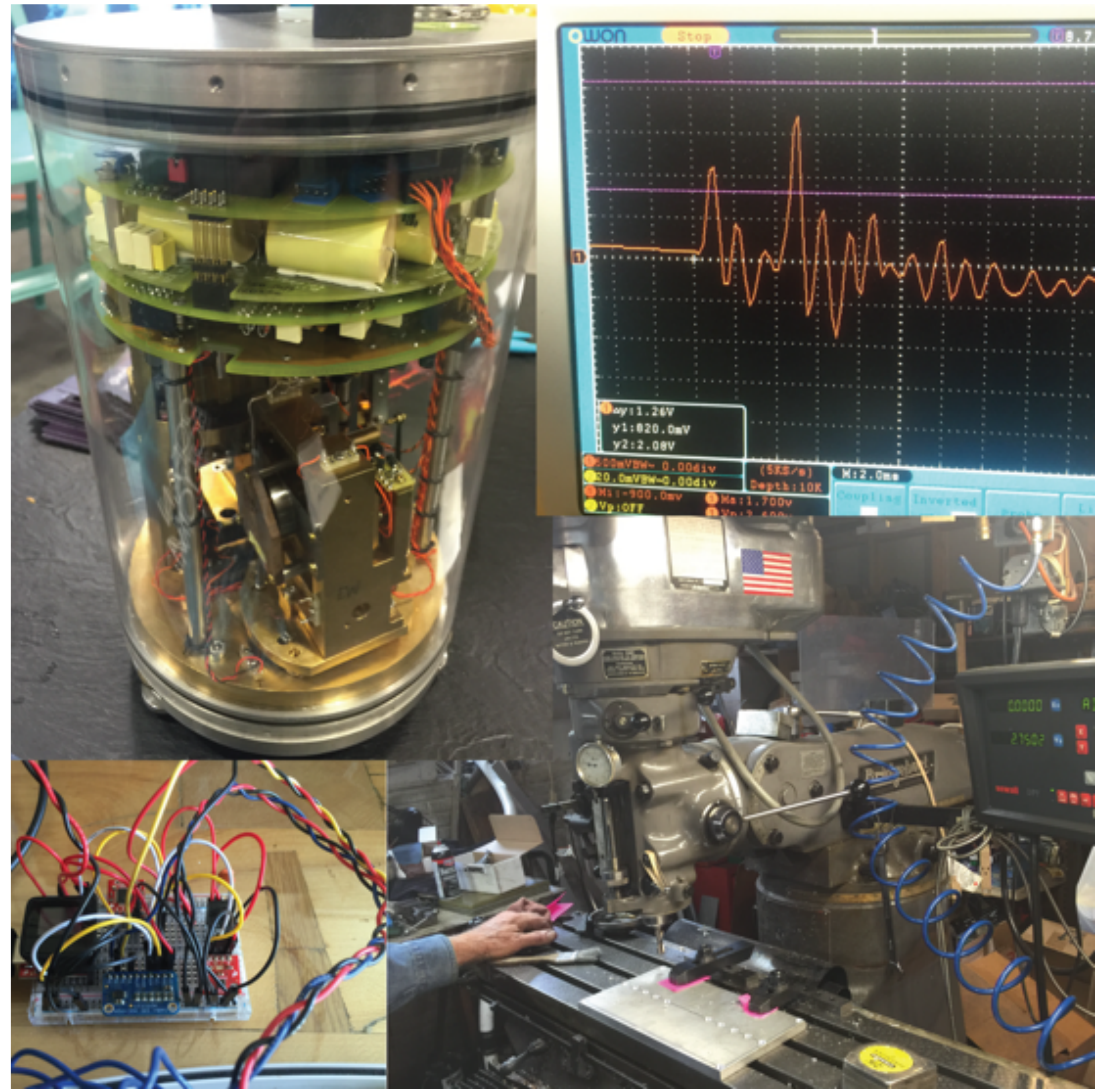
Digital Logic and Representation

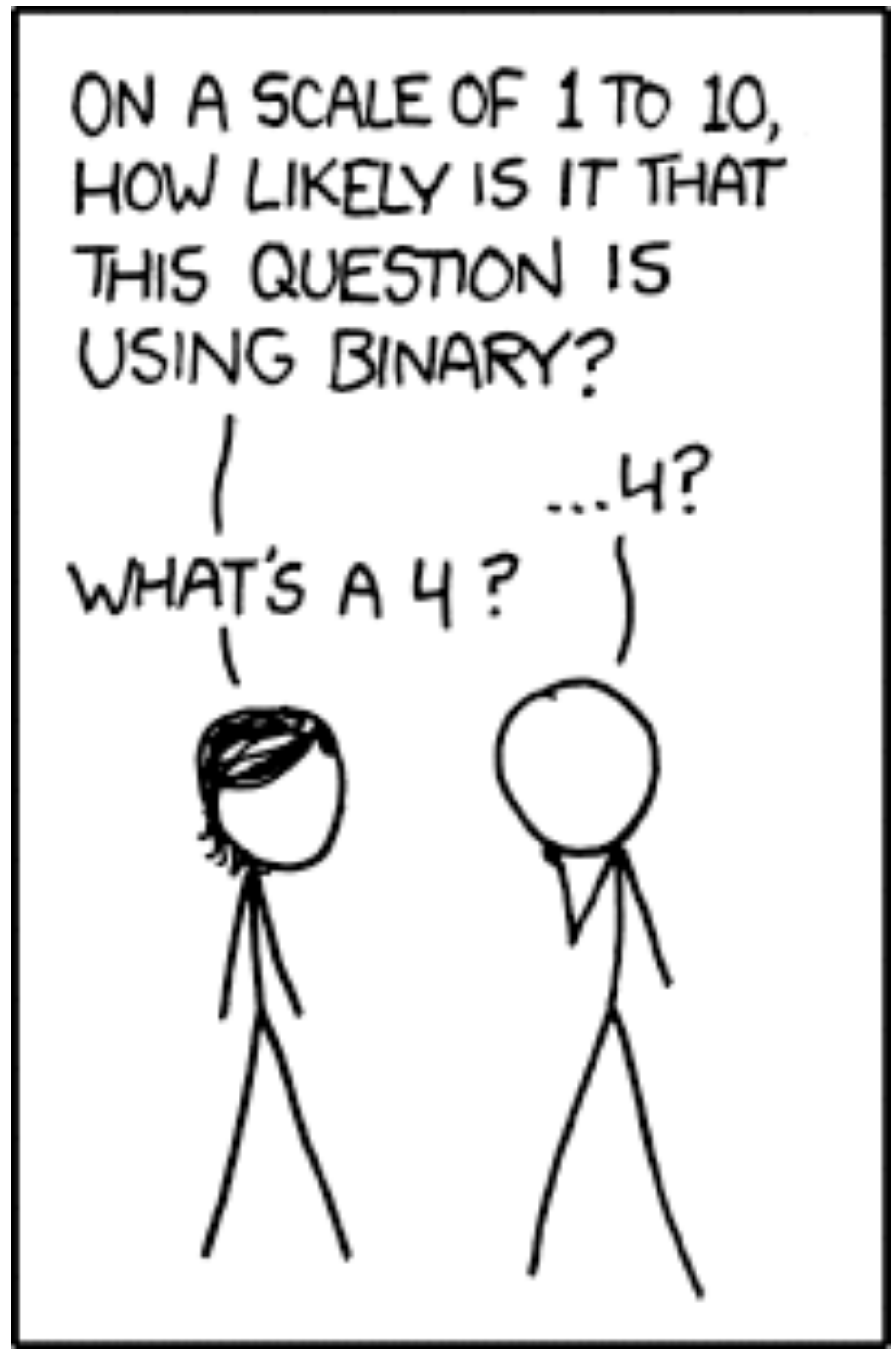
J.R. Leeman and C. Marone

Techniques of Geoscientific Experimentation

September 20, 2016



The way we think of and store numbers and do logic is much different than how computers do



XKCD.com

In digital logic there are two states

High

Low

Image: bobvila.com

In reality we have to define ranges of voltages that represent high and low states

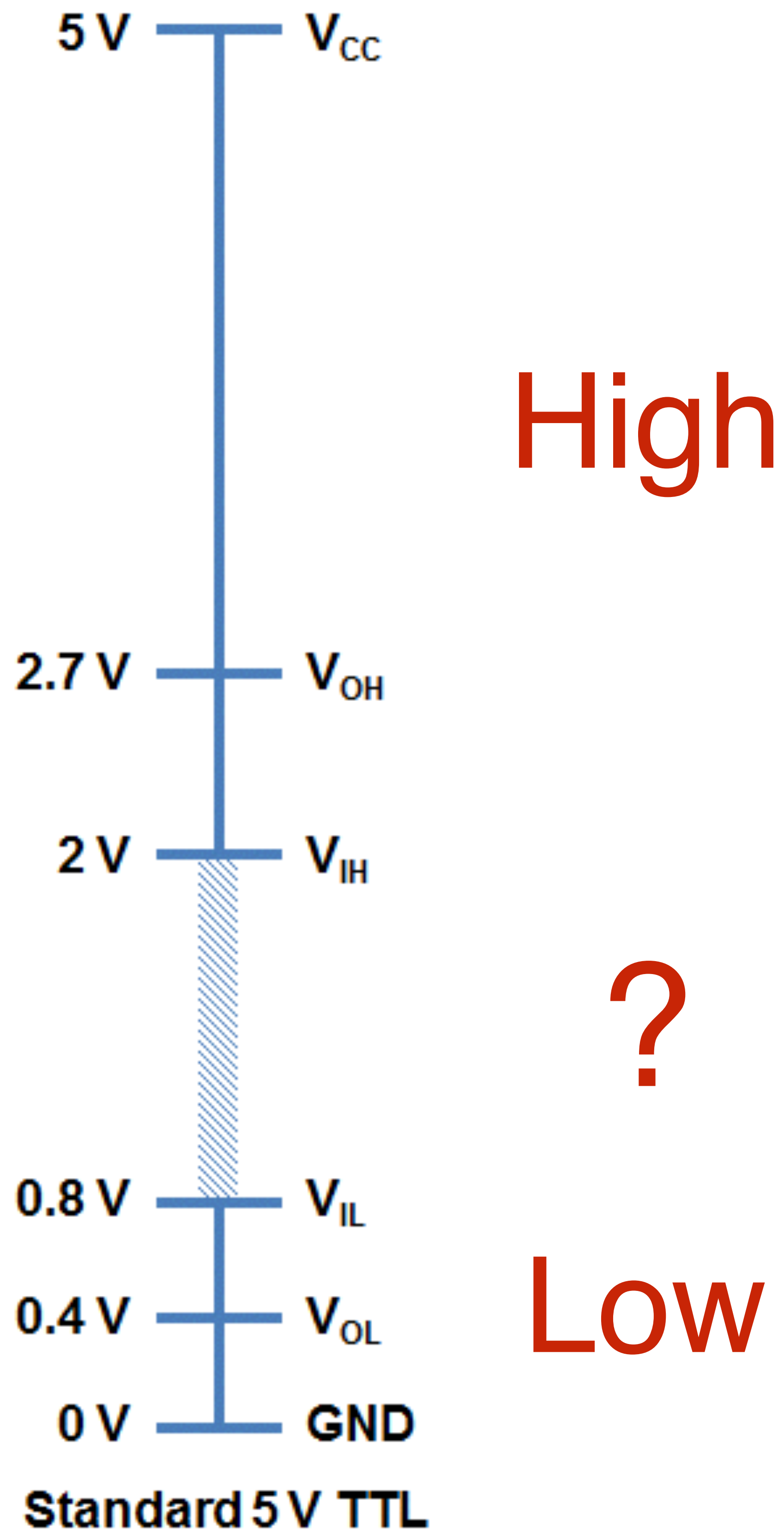


Image: sparkfun.com

In reality we have to define ranges of voltages that represent high and low states

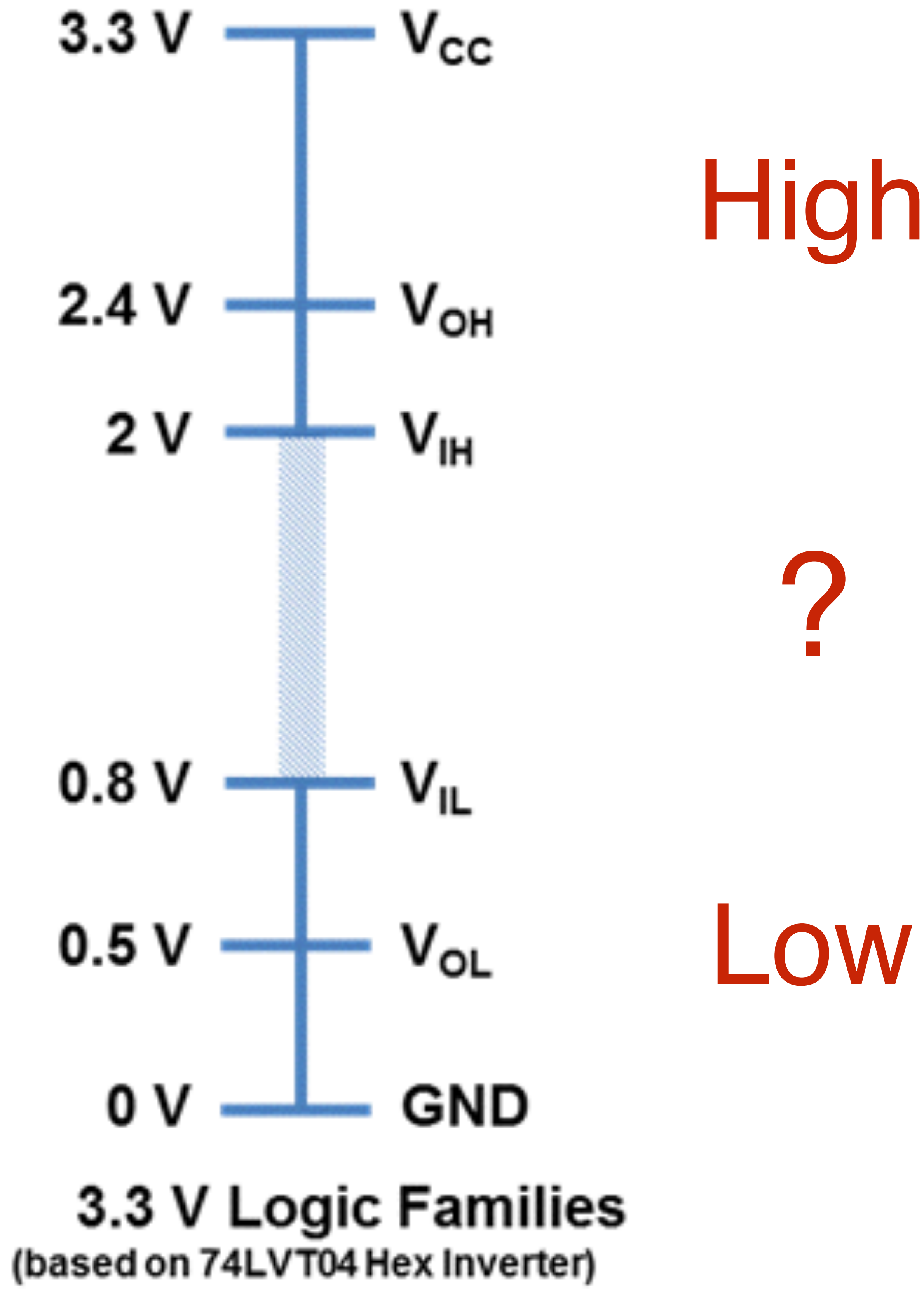
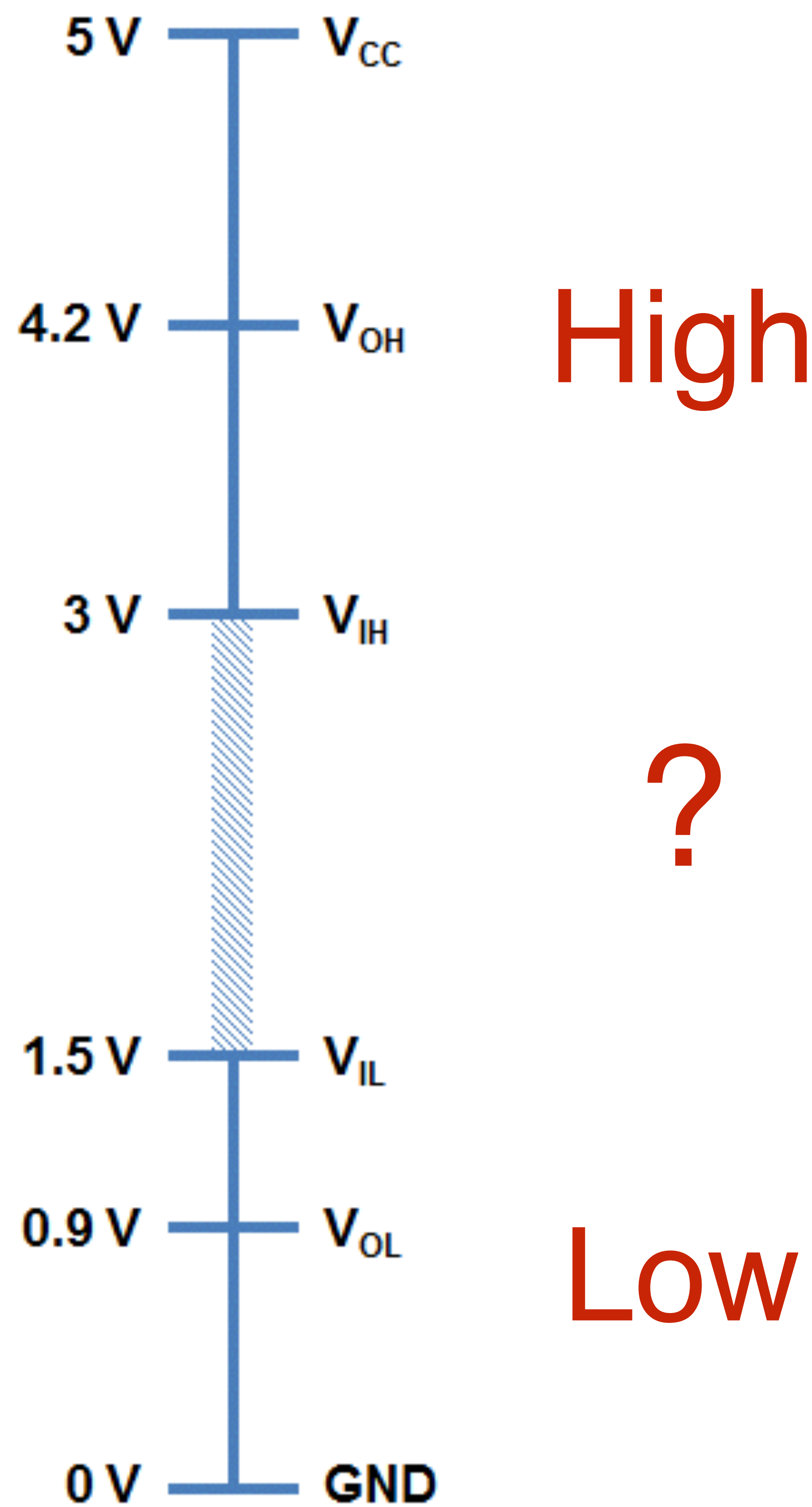


Image: sparkfun.com

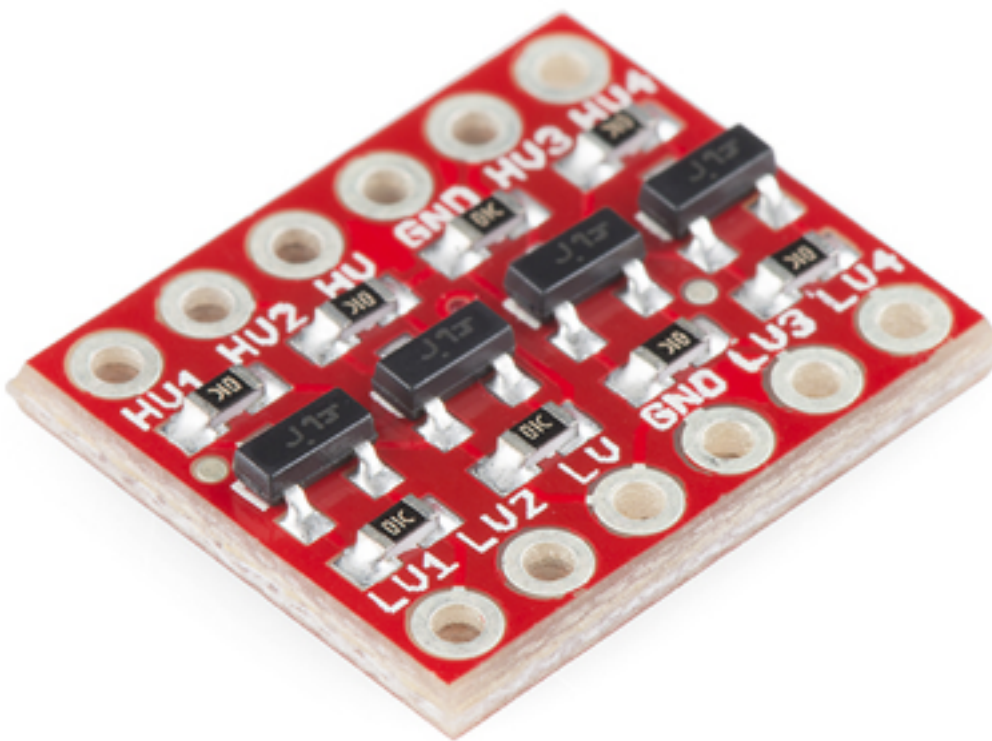
In reality we have to define ranges of voltages that represent high and low states

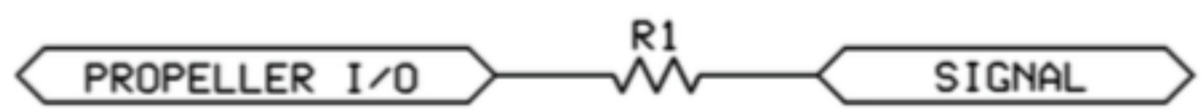


ATMega328
DC Characteristics

Image: sparkfun.com

Often we find the need to shift between these logic levels, which can be accomplished with a variety of techniques



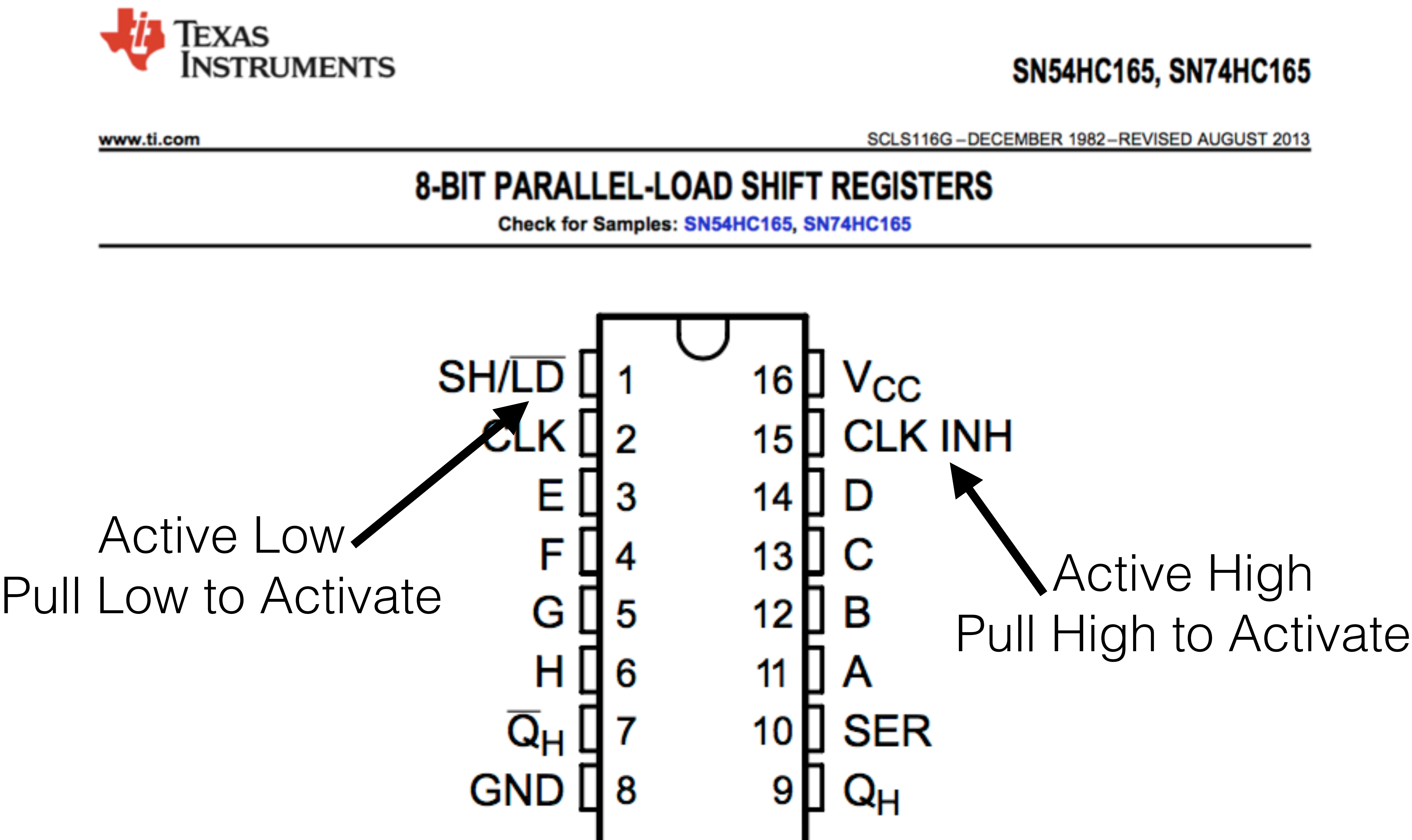


To choose a resistance value for R1, consider the following equation:

$$\textit{Minimum Resistance } R1 = \frac{|\textit{Signal Maximum } \Delta\, V|}{\textit{Maximum allowed ESD diode current}}$$

Images: sparkfun.com, parallax.com

Pins are often marked as active low or active high



There are a few digital logic operations that make up all of how we do computing with binary information

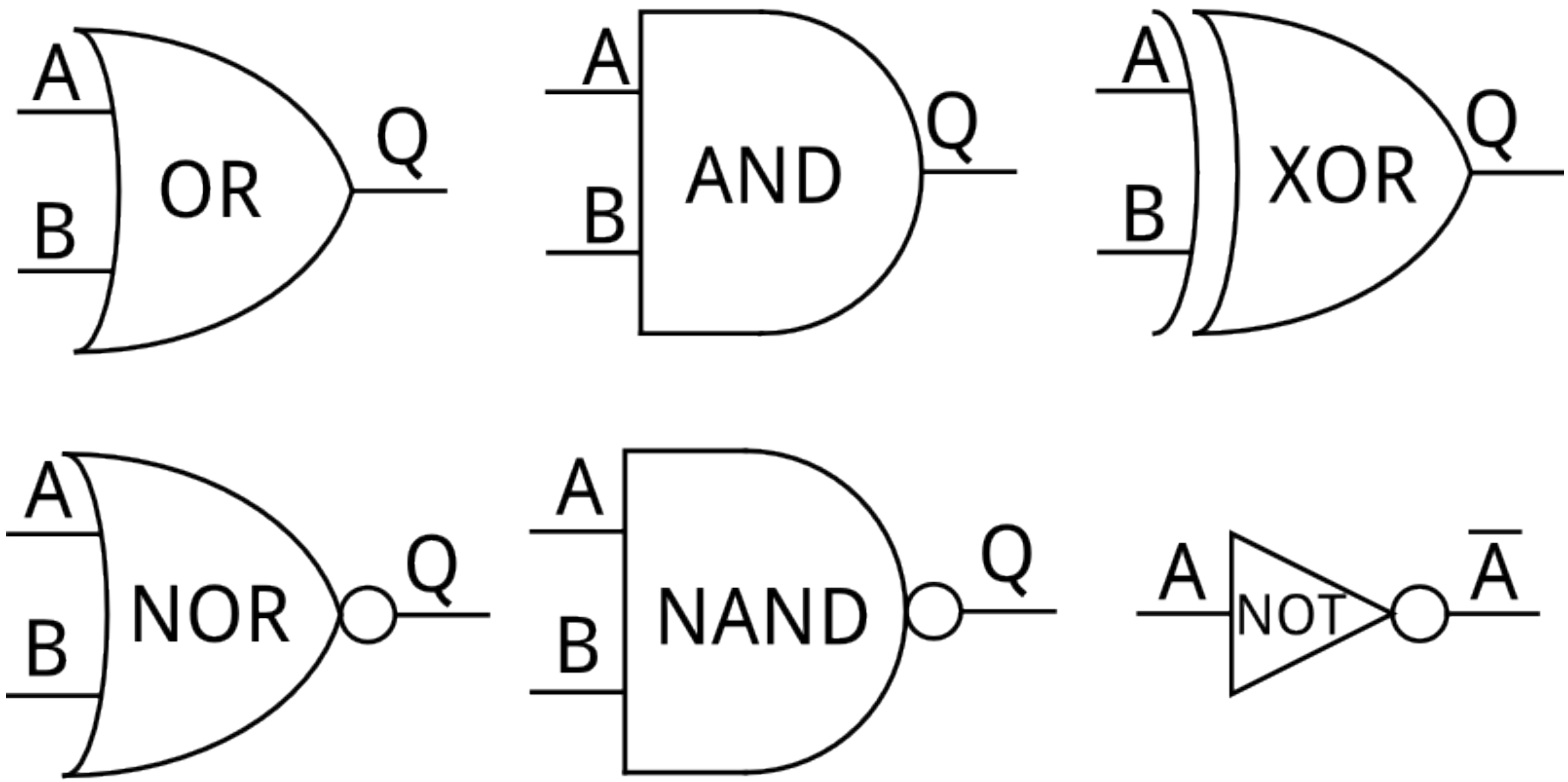


Image: sparkfun.com

We show the way each operation works with a truth table

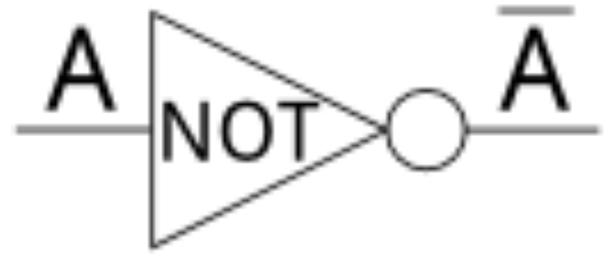
		A	
		0	1
B	0	Output	Output
	1	Output	Output

NOT gates invert the input

Written

$\bar{A}$

A'



A	
0	1
1	0

Image: sparkfun.com

AND gates are only true if both inputs are true

Written

$$AB$$

$$A \bullet B$$

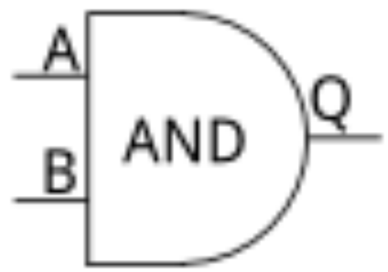
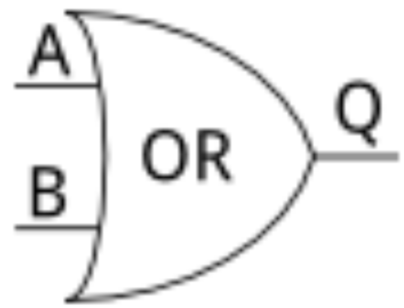
		A	
		0	1
B	0	0	0
	1	0	1

Image: sparkfun.com

OR gates are only true if either or both inputs are true

Written

$$A + B$$



		A	
		0	1
B	0	0	1
	1	1	1

Image: sparkfun.com

XOR gates are only true if either input is true

Written

$$A \oplus B$$

		A	
		0	1
B	0	0	1
	1	1	0

Image: sparkfun.com

NOR gates are only true if neither input is true

<u>Written</u>

$$\overline{(\mathrm{A}+\mathrm{B})}$$

$$(\mathrm{A}+\mathrm{B})'$$

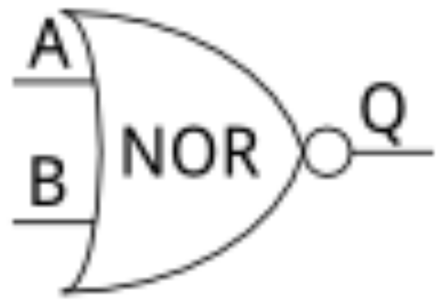
		A	
		0	1
B	0	1	0
	1	0	0

Image: <u>sparkfun.com</u>

NAND gates are true unless both inputs are true

Written

$$\overline{(\mathrm{AB})}$$

$$(\mathrm{A} \bullet \mathrm{B})'$$

$$(\mathrm{AB})'$$

		A	
		0	1
B	0	1	1
	1	1	0

Image: sparkfun.com

We can connect these logic gates together to perform calculations and other functions in a combinatorial logic circuit

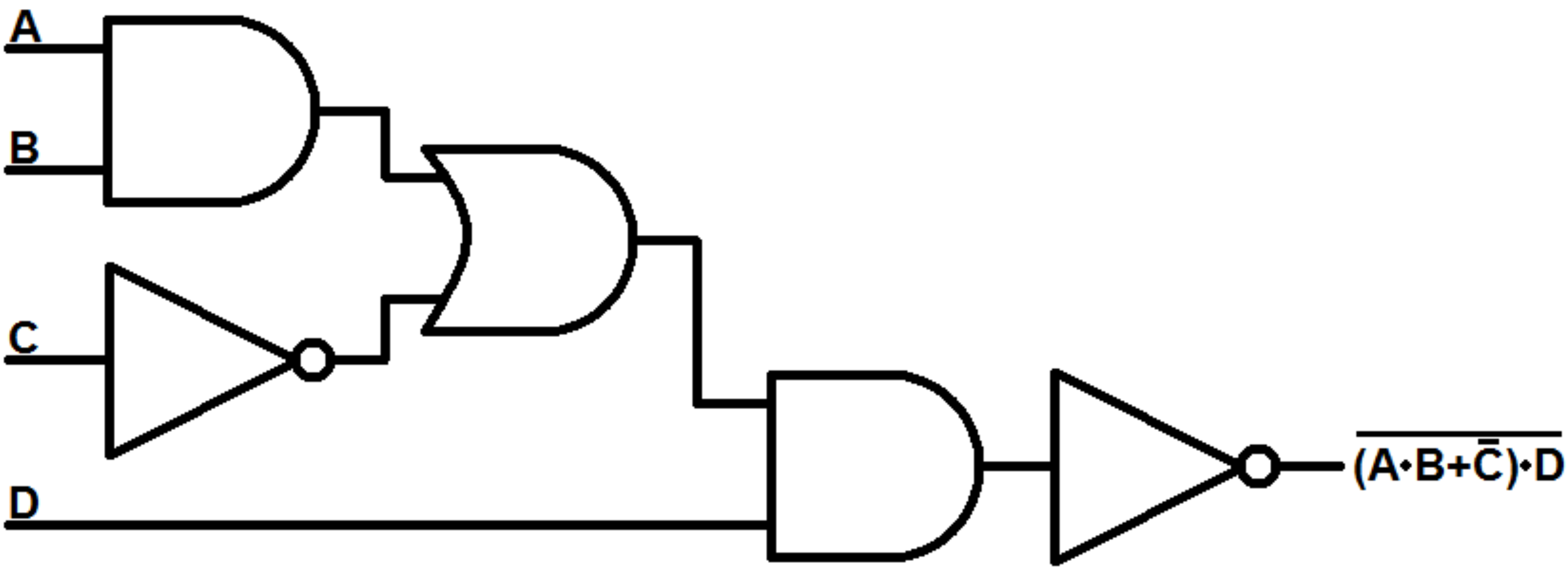
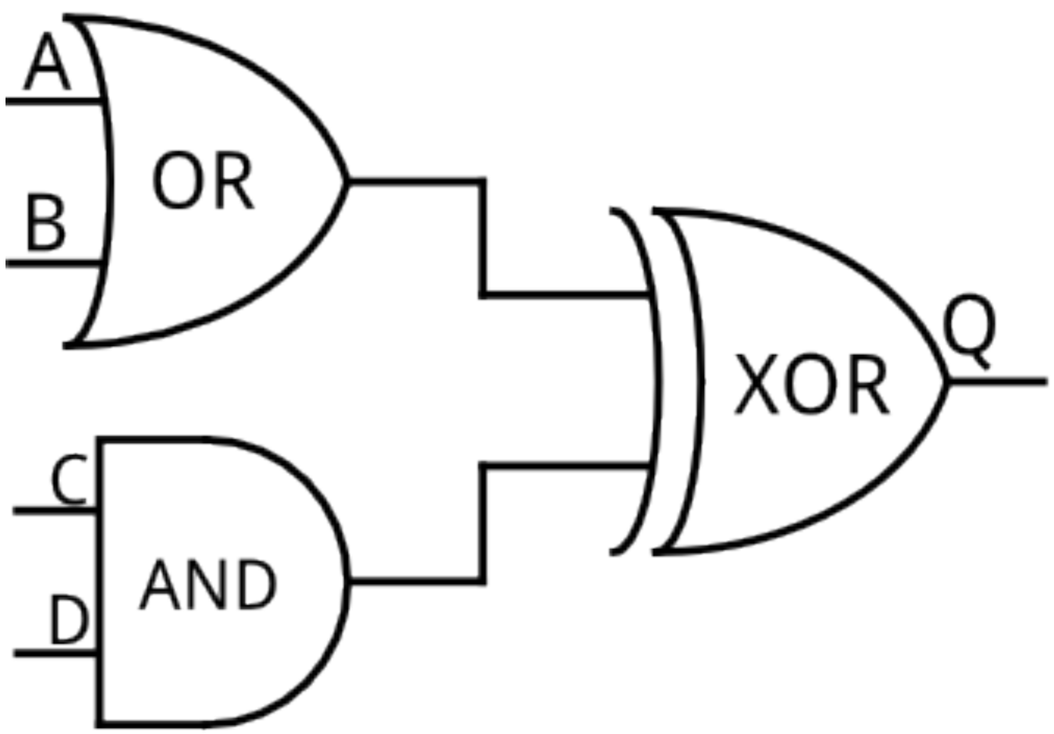


Image: sparkfun.com

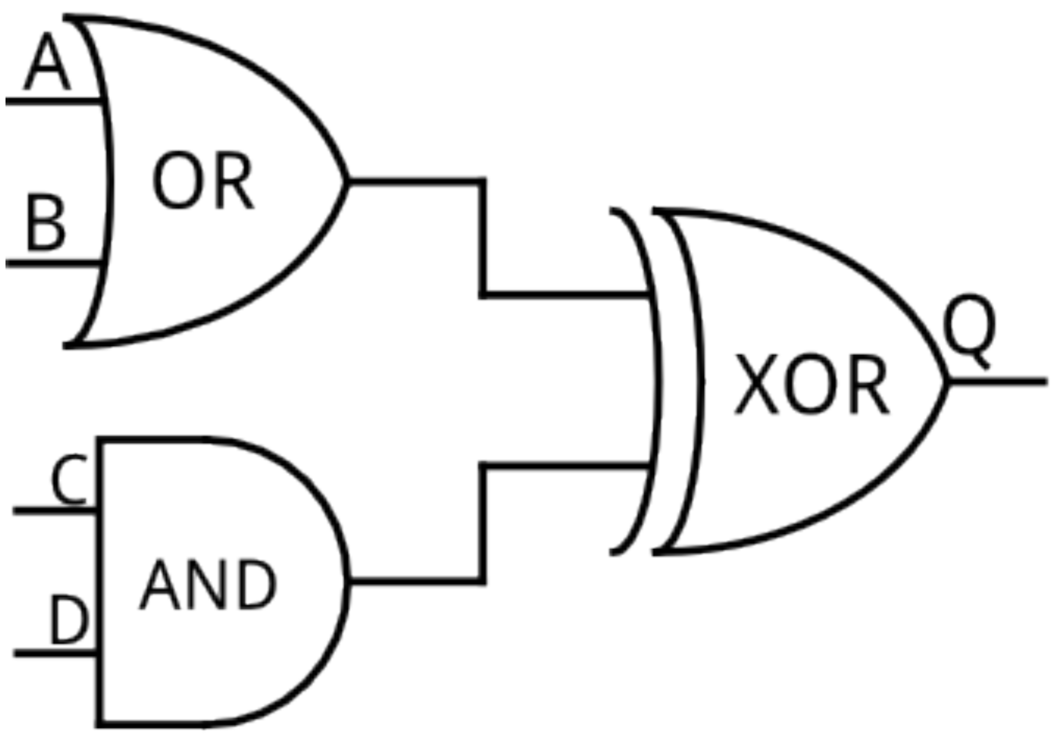
Activity: Fill out the truth table and write an expression for the output of this combinatorial circuit



		AB			
		00	01	10	11
CD	00				
	01				
	10				
	11				

Image: sparkfun.com

Activity: Fill out the truth table and write an expression for the output of this combinatorial circuit



		AB			
		00	01	10	11
CD	00	0	1	1	1
	01	0	1	1	1
	10	0	1	1	1
	11	1	0	0	0

Image: sparkfun.com

We can also make a sequential circuit that uses memory and (generally) a clock signal

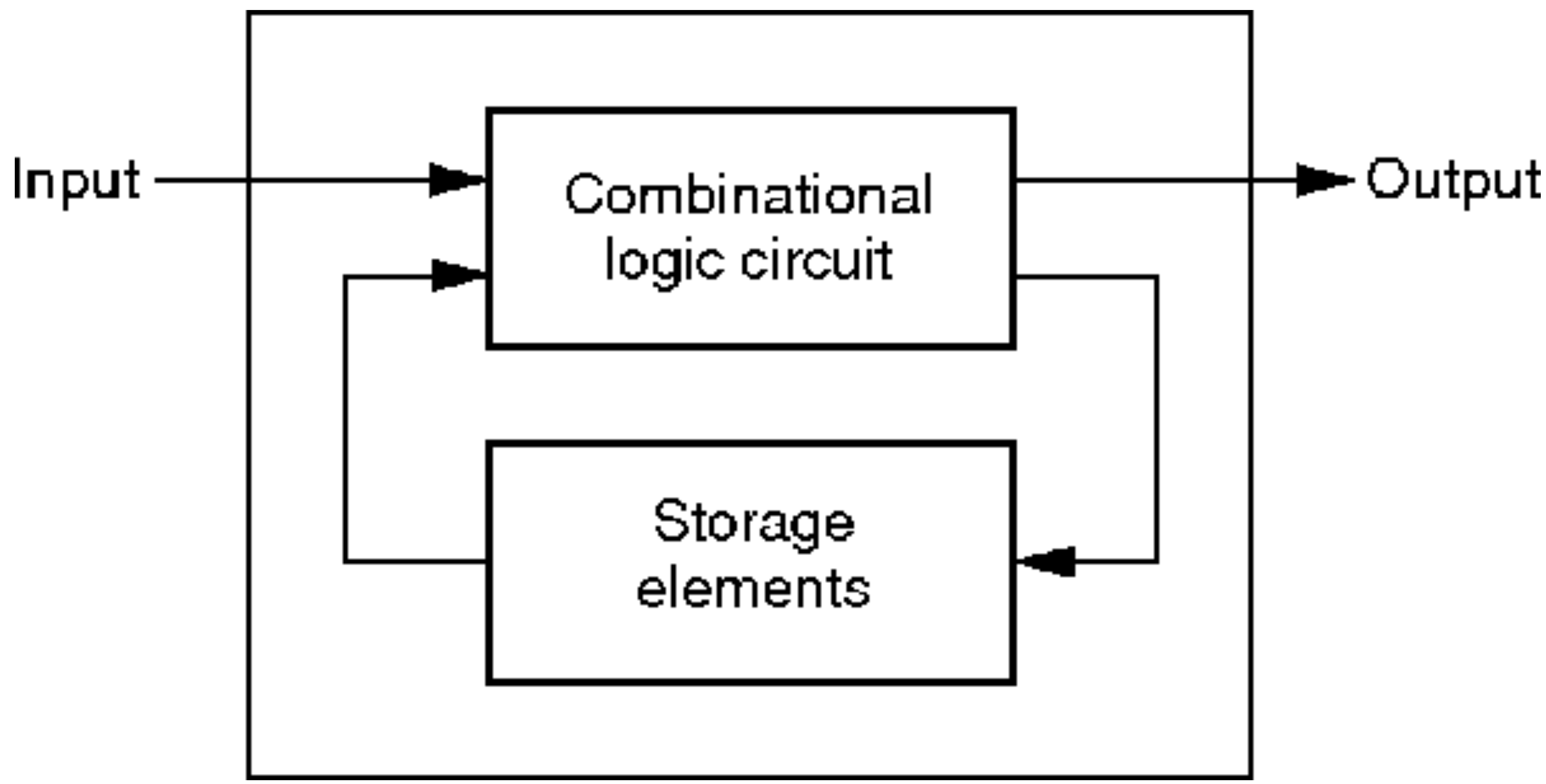


Image: c-jump.com

D type flip-flops latch the input to the outputs on a clock

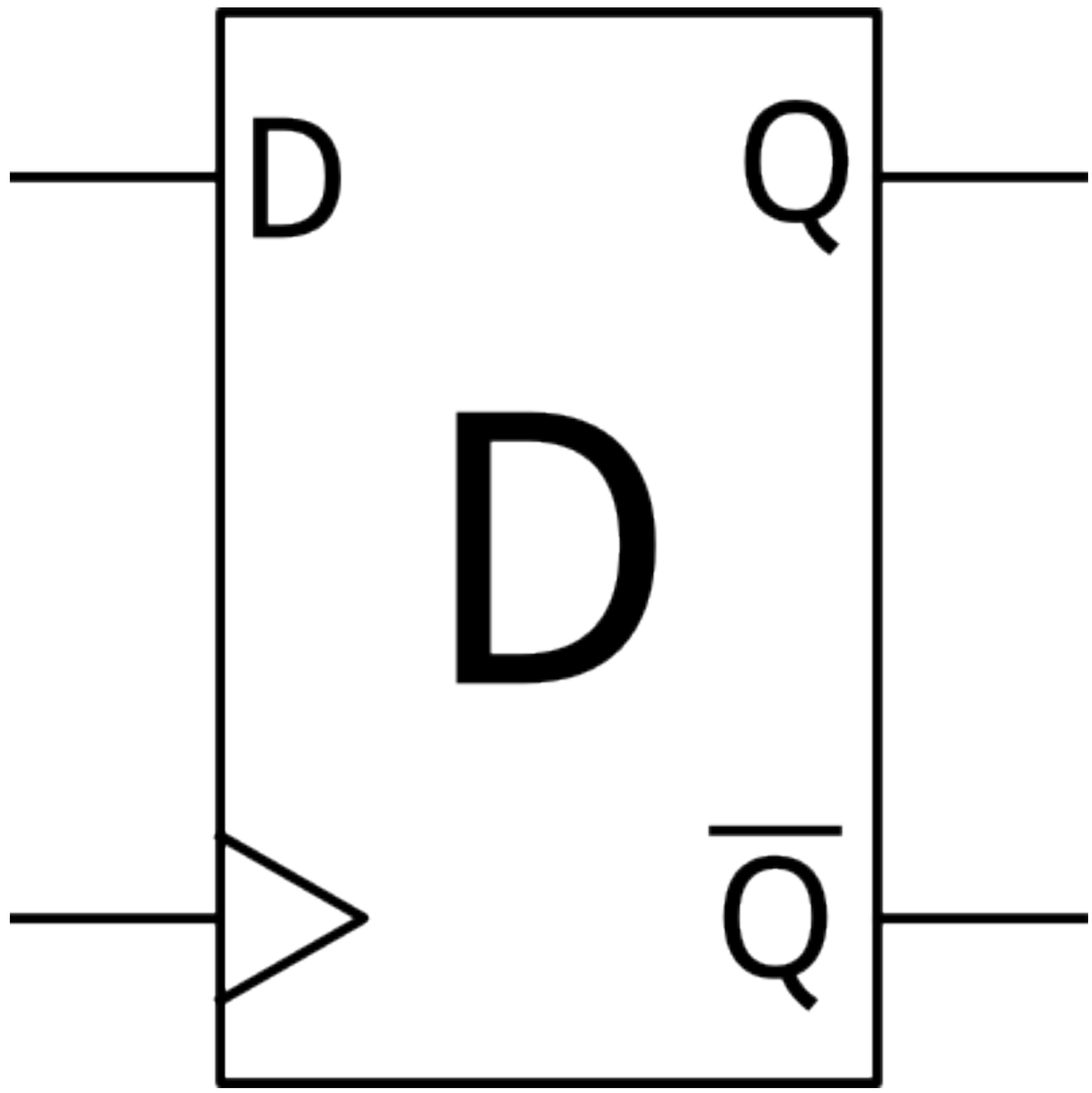


Image: sparkfun.com

T flip-flops toggle the output if T and do nothing if not T

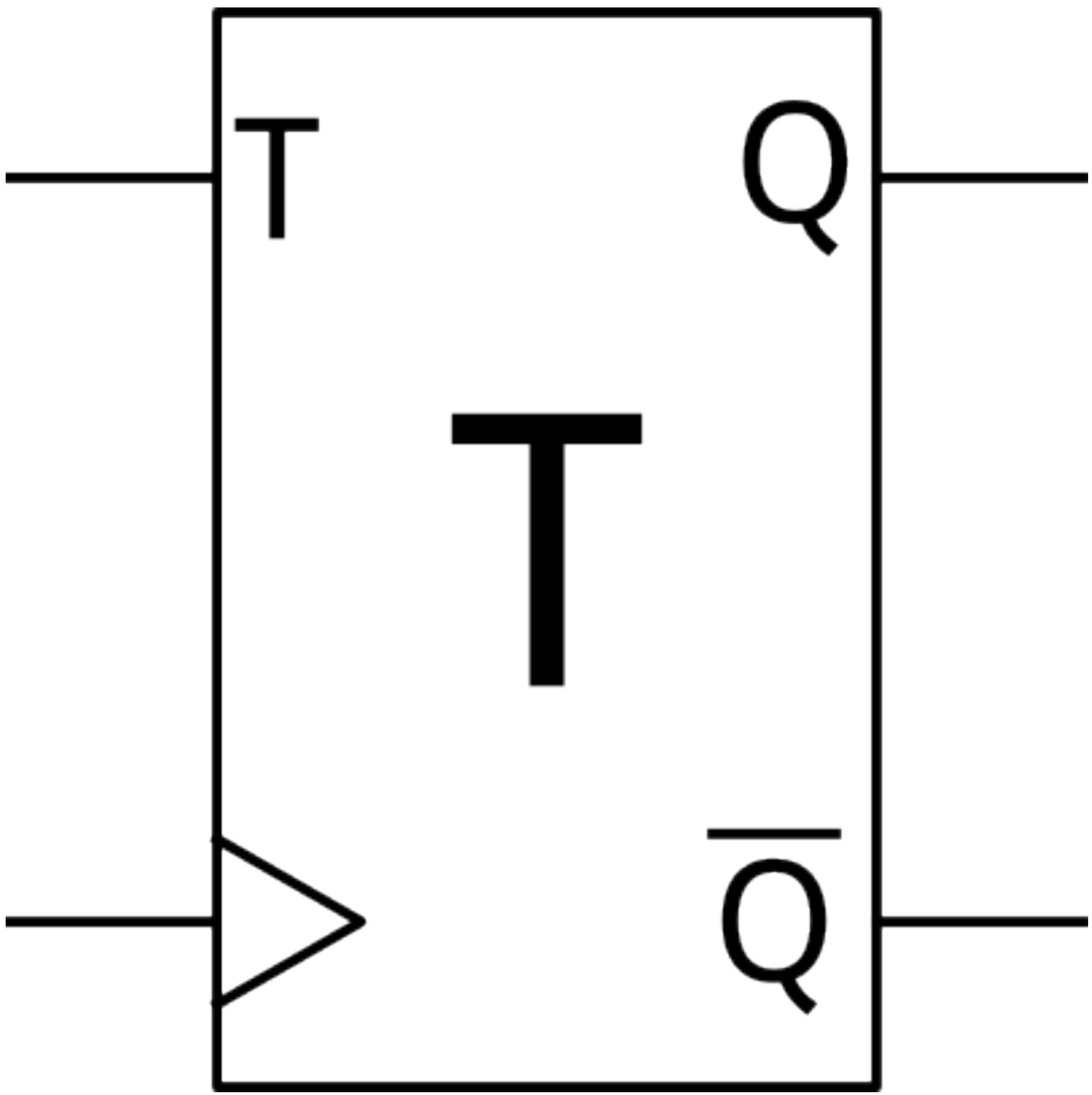
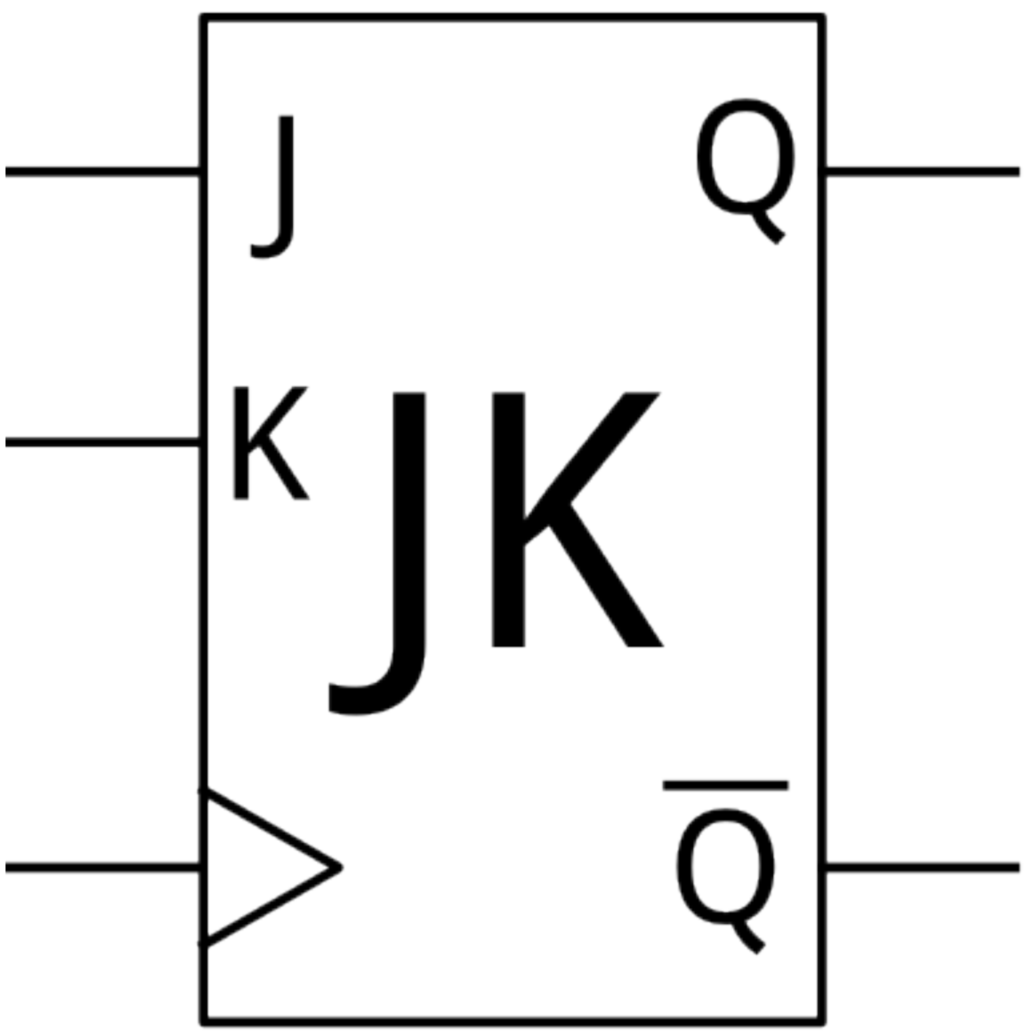


Image: sparkfun.com

JK flip-flops can set, clear, or toggle their outputs



JK	Q = 0	Q = 1
00	0	1
01	0	0
10	1	1
11	1	0

Image: sparkfun.com

We can perform logic operations in software as well

```
byte a = b01010101;
byte b = b10101010;
byte c;

c = a & b;  // bitwise AND-ing of a and b; the result is b00000000
c = a | b;  // bitwise OR-ing of a and b; the result is b11111111
c = a ^ b; // bitwise XOR-ing of a and b; the result is b11111111
c = ~a;  // bitwise complement of a; the result is b10101010
```

We use such operations to manipulate data when working with registers for example

```
byte a = b01010101;
byte b = b10101010;
byte c;

c = b00001111 & a; // clear the high nibble of a, but leave the low nibble alone.
                                // the result is b00000101.
c = b11110000 | a; // set the high nibble of a, but leave the low nibble alone.
                                // the result is b11110101.
c = b11110000 ^ a; // toggle all the bits in the high nibble of a.
                                // the result is b10100101.
```

We also often bit shift values to “roll” them

```
byte d = b11010110;
byte e = d>>2;  // right-shift d by two positions; e = b00110101
e = e<<3; // left-shift e by three positions; e = b10101000
```

Let’s learn how to translate binary numbers into base 10 representations

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
128	64	32	16	8	4	2	1

Let's learn how to translate binary numbers into base 10 representations

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
128	64	32	16	8	4	2	1
0	1	0	1	1	0	1	1

Let's learn how to translate binary numbers into base 10 representations

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
128	64	32	16	8	4	2	1
0	1	0	1	1	0	1	1
0	64	0	16	8	0	2	1

Let's learn how to translate binary numbers into base 10 representations

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
128	64	32	16	8	4	2	1
0	1	0	1	1	0	1	1
0	64	0	16	8	0	2	1
							91

The most significant bit can be first or last, we just have to agree and know what was done

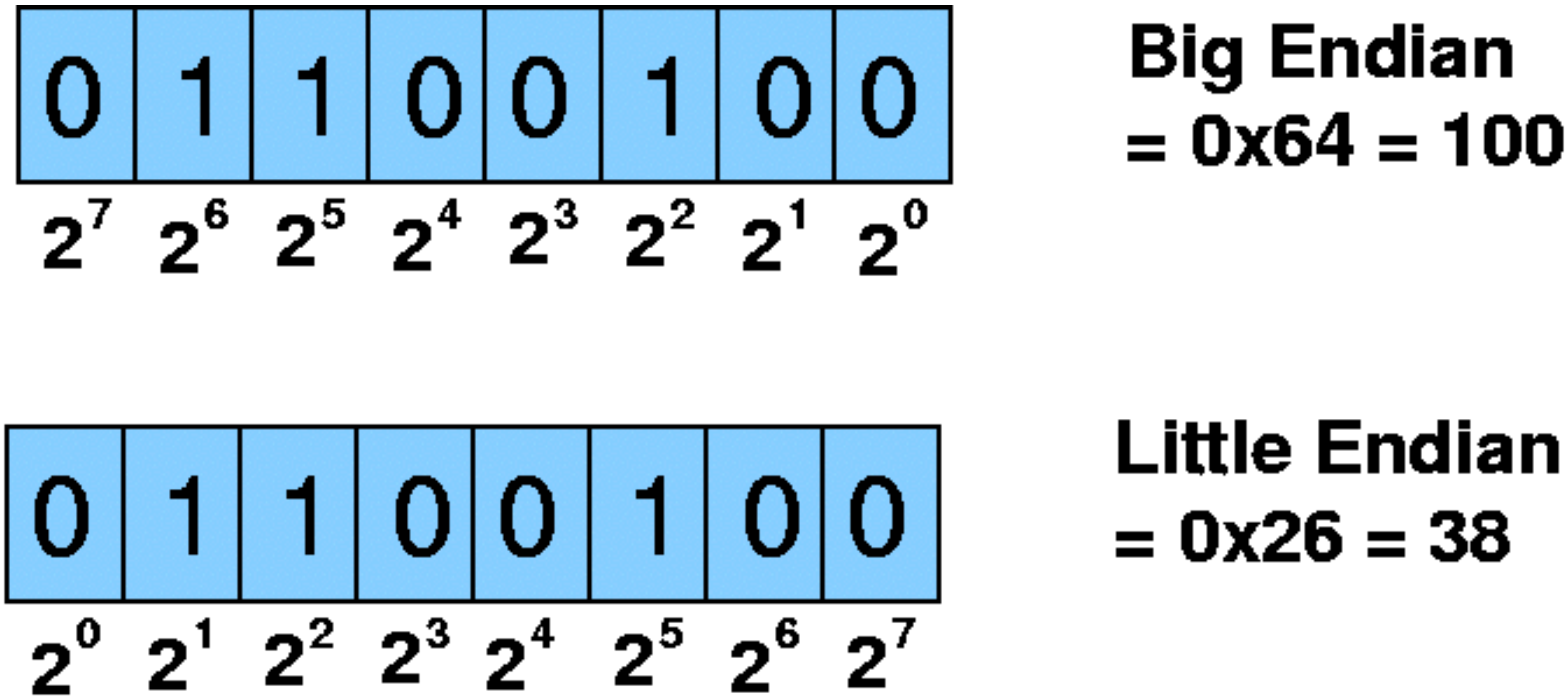
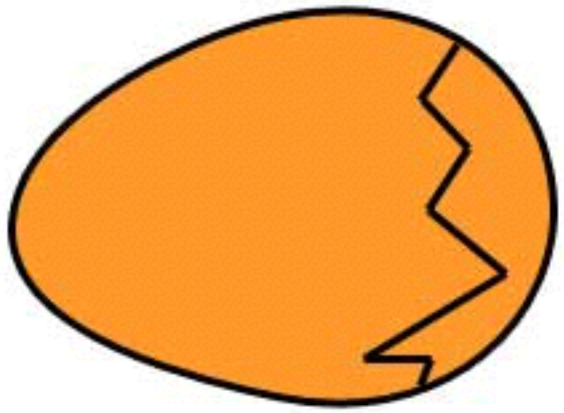
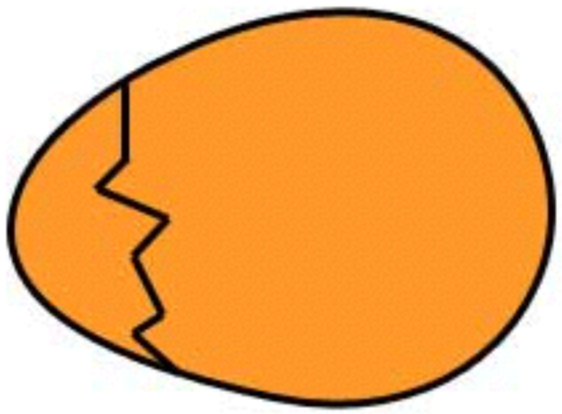


Image: users.cis.fiu.edu

Endianness gets its name from Swift's Gulliver's Travels



BIG ENDIAN - The way people always broke their eggs in the Lilliput land



LITTLE ENDIAN - The way the king then ordered the people to break their eggs

Image: flickeringtubelight.com

We've explored base 10 and base 2, but what if we want more than 0-9? Base 16!

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Image: sparkfun.com

Counting is a bit strange since we're used to base 10

Decimal	Hexadecimal	...	Decimal	Hexadecimal
0	0		8	8
1	1		9	9
2	2		10	A
3	3		11	B
4	4		12	C
5	5		13	D
6	6		14	E
7	7		15	F

Image: sparkfun.com

Counting is a bit strange since we're used to base 10

Decimal	Hexadecimal	...	Decimal	Hexadecimal
16	10		24	18
17	11		25	19
18	12		26	1A
19	13		27	1B
20	14		28	1C
21	15		29	1D
22	16		30	1E
23	17		31	1F

Image: sparkfun.com

Let's convert the decimal number 48879 to hex

48879/16 = 3054 R 15 > F

3054/16 = 190 R 14 > EF

190/16 = 11 R 14 > EEF

11/16 = 0 R 11 > BEEF

Converting to binary is a powers of 16 problem

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
16^7	16^6	16^5	16^4	16^3	16^2	16^1	16^0
268435456	16777216	1048576	65536	4096	256	16	1

Converting from binary is done by grouping into bunches of 4

Binary: 1011111011101111

Binary: 1011 | 1110 | 1110 | 1111

Decimal: 11 | 14 | 14 | 15

HEX: B | E | E | F

You'll see all of these formats written in a variety of ways

Hex

0xBEEF
#FF7454
%20
\x1B
&#BD
0hBEEF
BEEF_{16}
BEEF_{hex}

Binary

0b01110100
%01110100
%0111_0100

Two's complement lets us represent negative numbers

8-bit	
uint8_t	0 to 255
int8_t	-128 to 127

Converting to two's compliment is "simple"

1. Write out the number in binary
2. Invert all of the digits
3. Add one to the result

Write the number -42

1. Write out the number in binary
2. Invert all of the digits
3. Add one to the result

$$42 = 2 + 8 + 32$$

0010_1010

Write the number -42

1. Write out the number in binary
2. Invert all of the digits
3. Add one to the result

0010_1010

1101_0101

Write the number -42

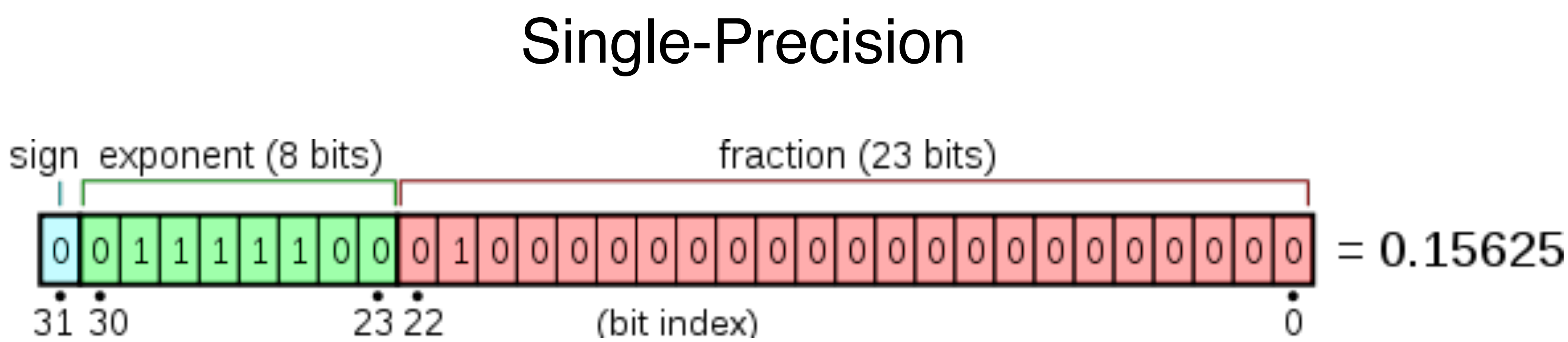
1. Write out the number in binary
2. Invert all of the digits
3. Add one to the result

1101_0101

↓

1101_0110

Floating point is expensive, but useful



sign * 2^{exponent} * mantissa

In this example:

- $\text{sign} = b_{31} = 0$
- $(-1)^{\text{sign}} = (-1)^0 = +1 \in \{-1, +1\}$
- $e = b_{30} b_{29} \dots b_{23} = \sum_{i=0}^{7} b_{23+i} 2^{+i} = 124 \in \{1, \dots, (2^8 - 1) - 1\} = \{1, \dots, 254\}$
- $2^{(e-127)} = 2^{124-127} = 2^{-3} \in \{2^{-126}, \dots, 2^{127}\}$
- $1.b_{22} b_{21} \dots b_0 = 1 + \sum_{i=1}^{23} b_{23-i} 2^{-i} = 1 + 1 \cdot 2^{-2} = 1.25 \in \{1, 1 + 2^{-23}, \dots, 2 - 2^{-23}\} \subset [1; 2 - 2^{-23}] \subset [1; 2)$

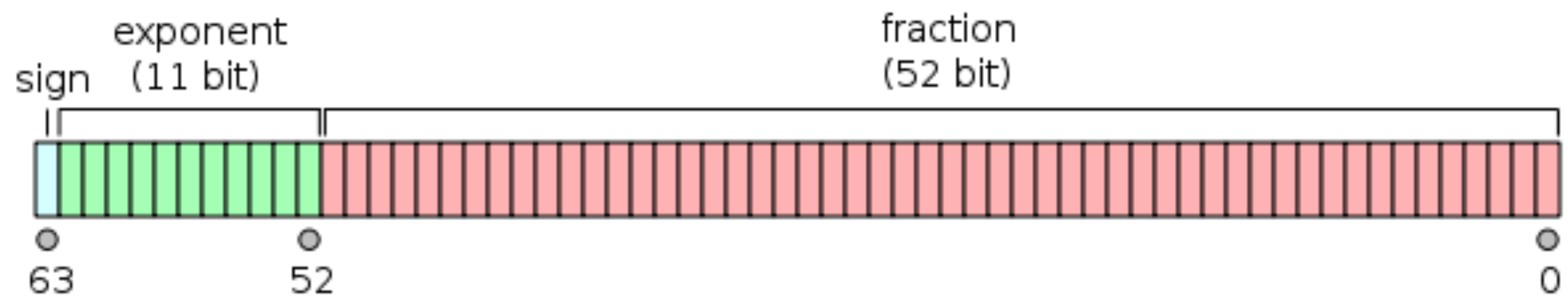
thus:

- $\text{value} = (+1) \times 1.25 \times 2^{-3} = +0.15625$

Image: wikipedia.com

Not all representations are exact and can accumulate error

Double-Precision



Decimal Number: 0.1
Single Precision: 0x3DCCCCCD
Cast to Double: 0.10000000149011612

1e6 * 0.1 (cast to double) = 100000.00149011612

Image: wikipedia.com

Assignment: Digital Representation

DUE: 9/27/16