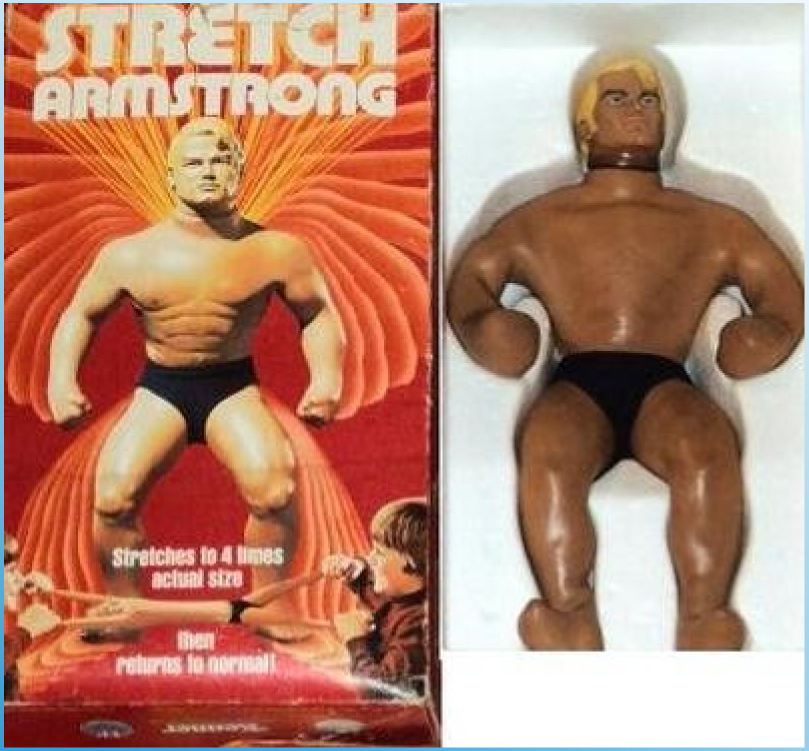
What's Inside

Lecture 2

- **Programming languages**, such as Java are designed for *theoretical Turing Machines*
- However, they also may have the *physical nature of the machine in mind*.
- We need to know how these things work under the hood!

Motivation

- WHAT MAKES IT SO STRETCHY?!



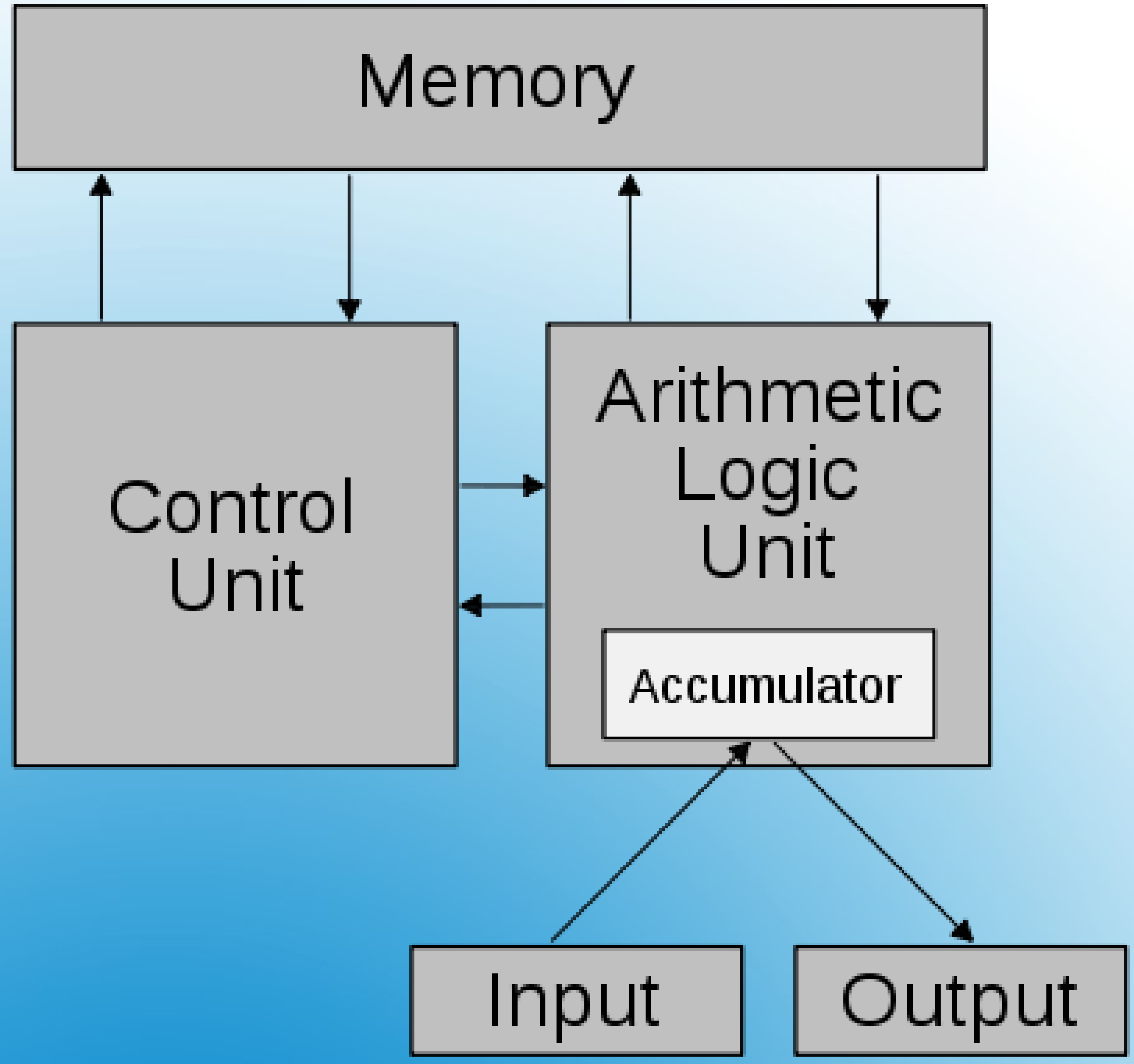
Motivation II

- WHAT MAKES IT SO BOUNCY?!

Motivation III

- Computer Architecture
  - How these machines are built
  - What the components do
- Dataflow
  - How are numbers represented?
  - How are letters represented?
  - Colors? Images? Videos?
- Control
  - Mathematical Operations

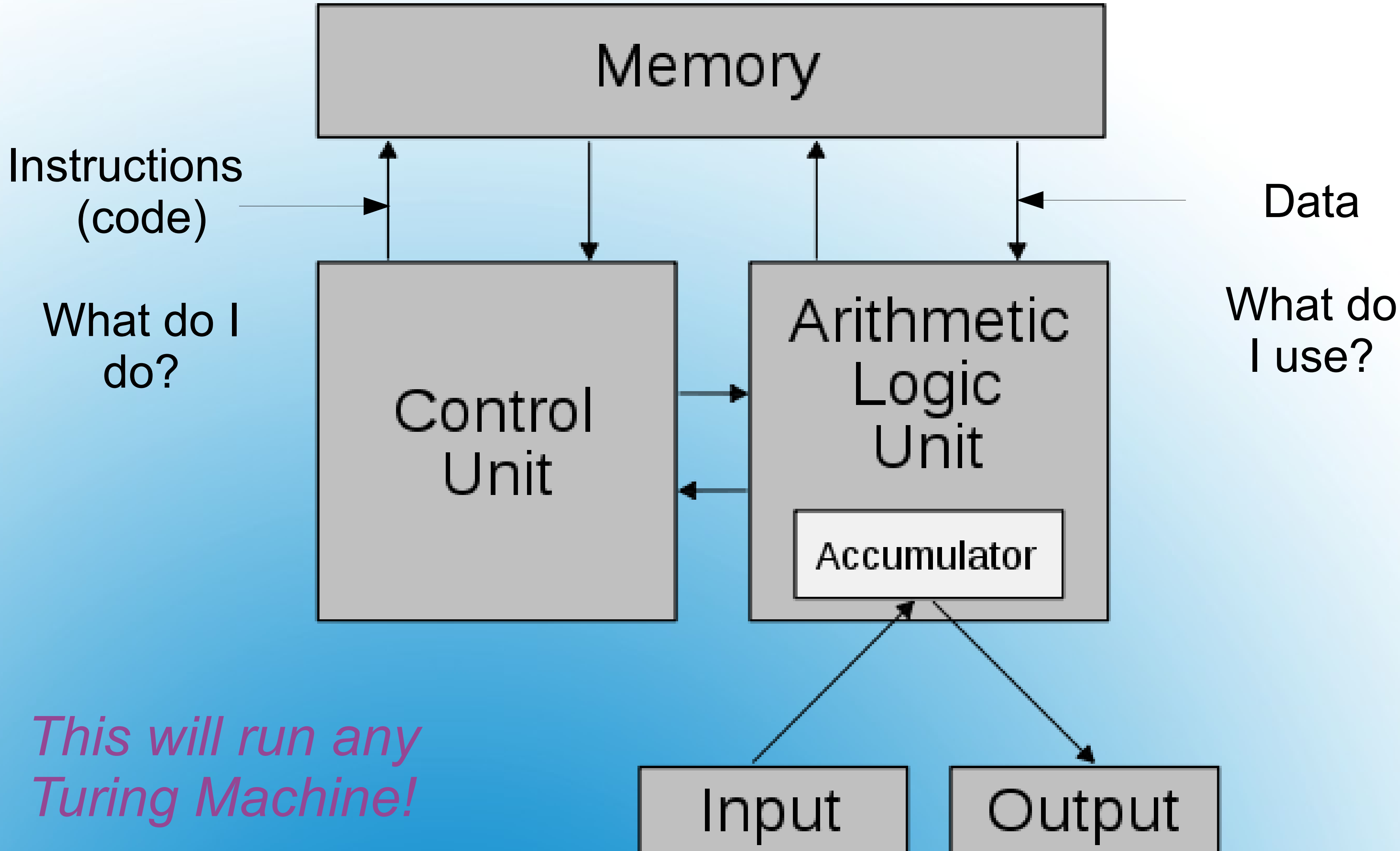
Outline



Computer Architecture

John Von Neumann

- **John Von Neumann** (1903-1957)
- Mathematician and nuclear physicist.
- Worked on the hydrogen bomb
- Collaborated with Eckert and Mauchly (ENIAC)
- The *Von Neumann architecture* came from a 1945 paper about the upcoming EDVAC
  - Eckert and Mauchly's names were left out :(



This will run any Turing Machine!

Stored-Program Computer

- The **memory** is a storage device.
  - Much like a set of bins
- This memory can read or store information.
- Memory can be considered *sequential access* and *random access*.

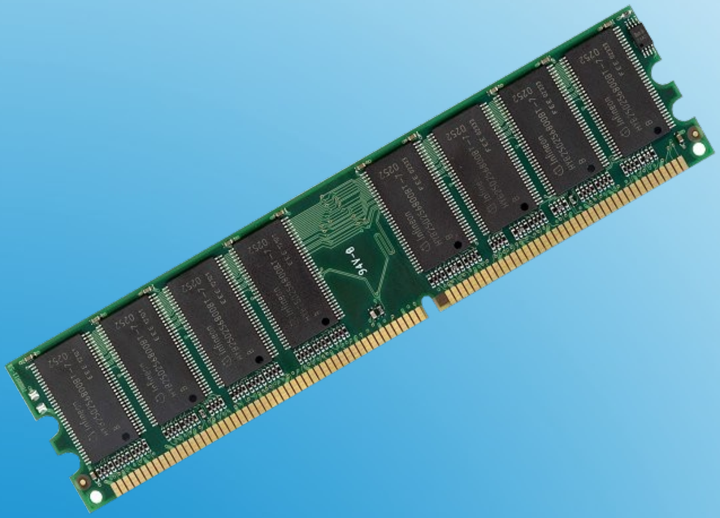
Memory

- There is *sequential memory*.
  - Like going through a filing cabinet a file at a time

Sequential Memory

Random Access Memory

- What we need here is *random access memory*.
  - Retrieving/Storing anything at any time regardless of its location.
  - Sequential memory can be random access...
    - As long as you are willing to wait!



- Memory stores two things
  - Code (Instructions)
  - Data (Information)
- The *control unit* reads instructions, and executes them.
- It also chooses the next instruction to be executed.
  - It may choose between two instructions
  - An earlier instruction might mean a loop!

Control Unit

- The last component is the unit that carries out the instruction. (**ALU**)
  - It implements basic logic and mathematics
  - Examples: Add, Subtract, Multiply, Compare

Σ $x \geq y$

ϕ Π $x + y$

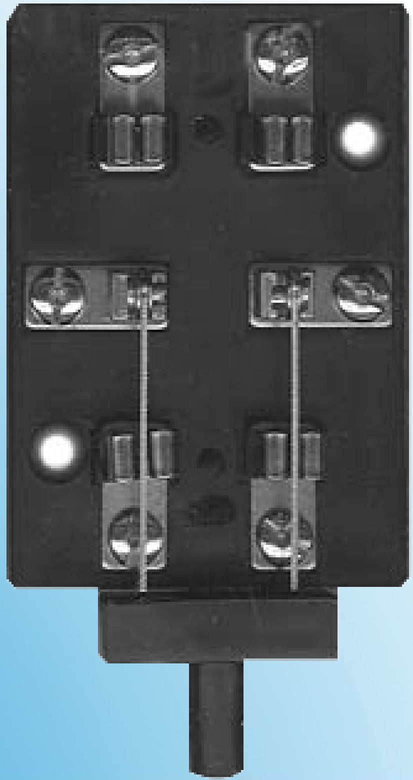
Arithmetic Logic Unit

Running a Program

- You may wonder... how can this memory hold everything we know a computer can manipulate?
  - Books?
  - Images?
  - Videos?
  - Sounds?
- The answer is a very common mechanical device.

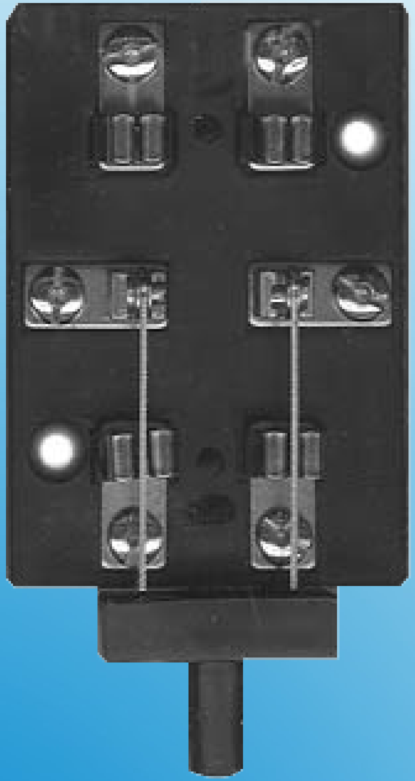
Representing Data

- This is how we will represent any piece of information!
  - The typical state-of-the-art "switch" is made up of *transistors*.
  - Hard drives use the polarity of a small magnet.
  - CDs use the difference between a flat region and a small hole measured by a laser.



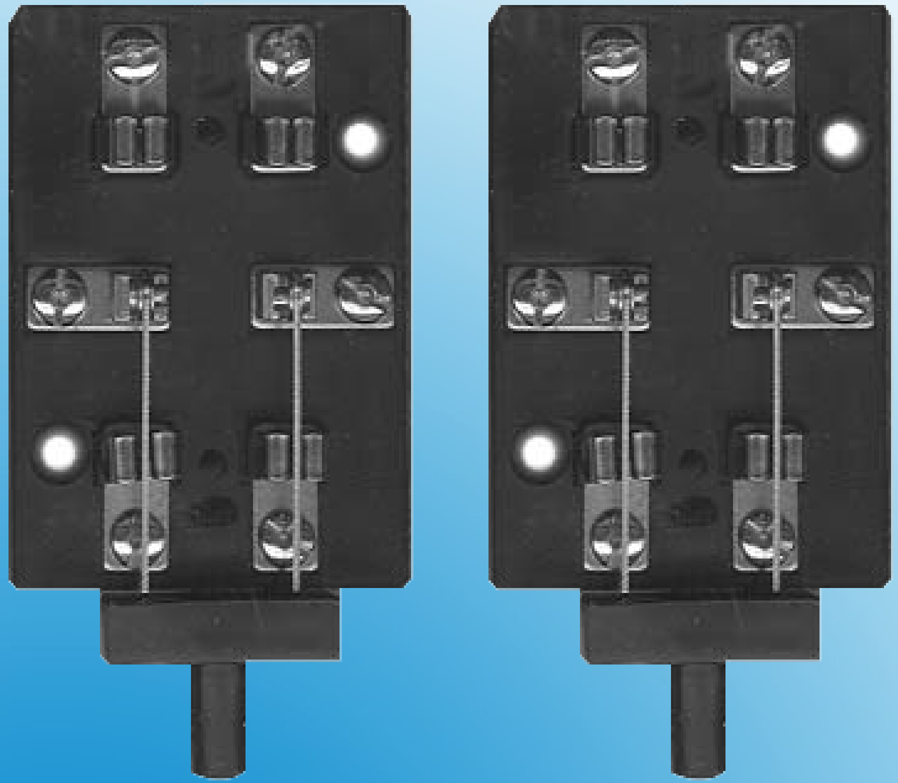
Switches

- This single unit of information is called a *bit*.
- How many different values can this represent?



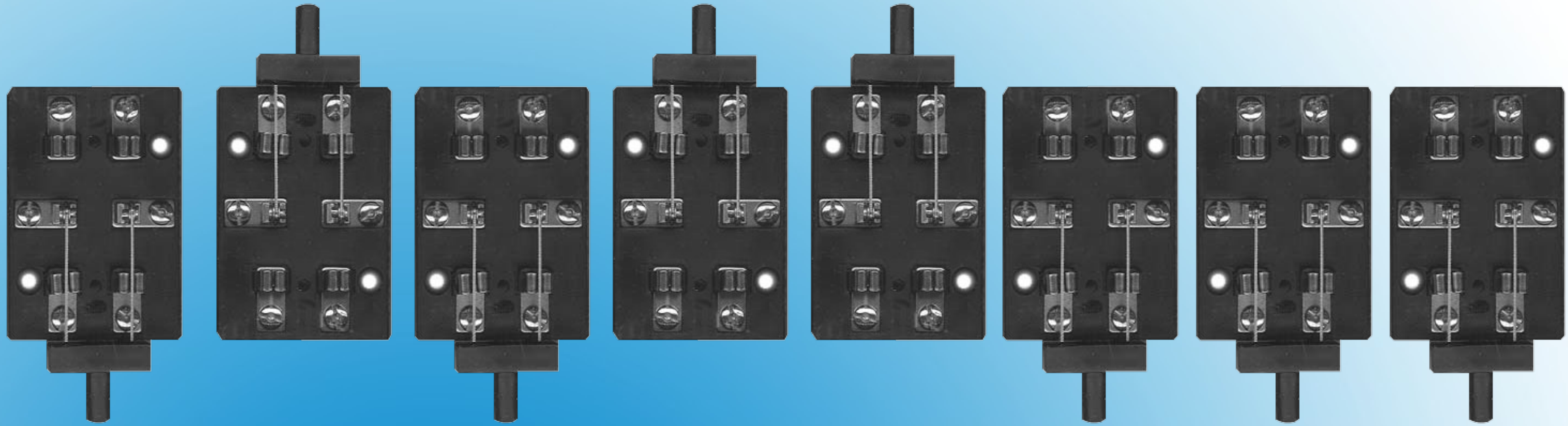
The Bit

- How many different values can this represent?



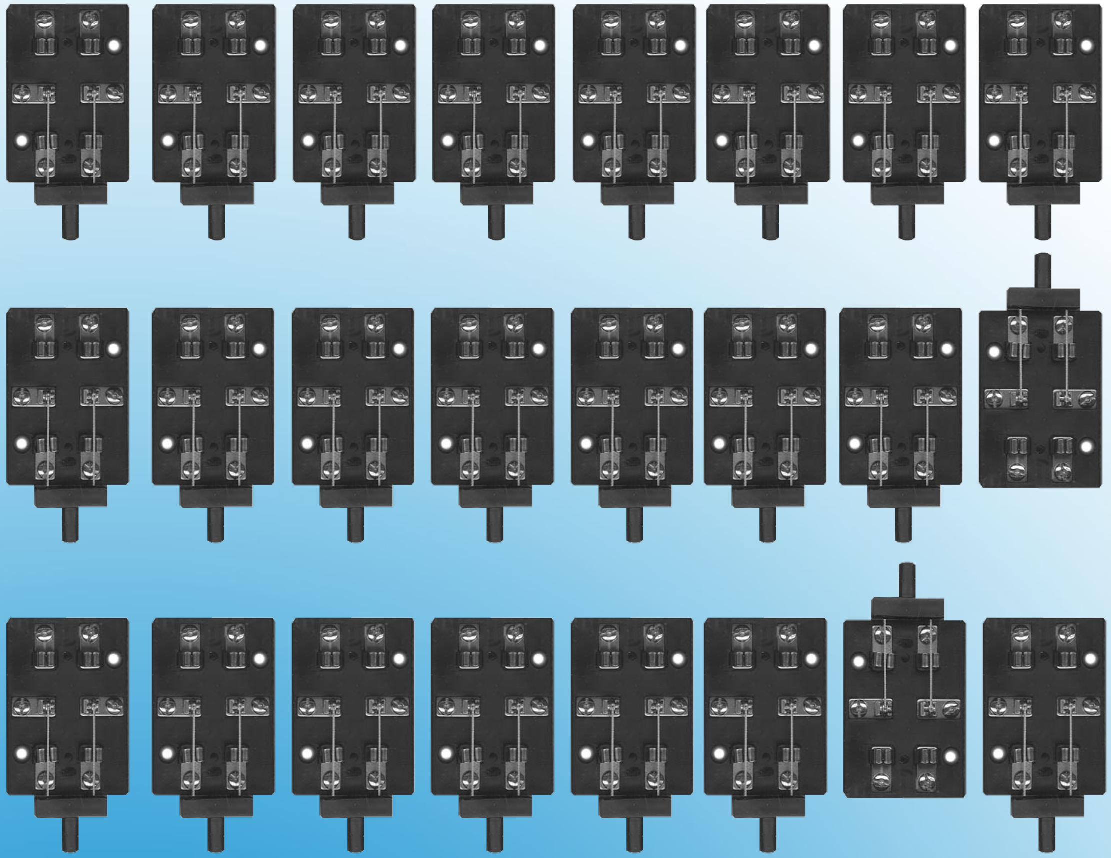
Switches

- With 8 switches we can represent how many values?
- The usage of 8 switches is actually the smallest unit of information.
  - It is called a *byte*.



The Byte

- 0
- 1
- 2
- ...



Representing Numbers

- Because of these *switches*, computers use what is called **binary encoding**.

- Normally we use **decimal** encoding... (base 10)

$$145 = (1)100 + (4)10 + (5)1 = (1)10^2 + (4)10^1 + (5)10^0$$

- In binary (base 2), 145 is:

$$10010001 = (1)2^0 + (0)2^1 + (0)2^2 + (0)2^3 + (1)2^4 + (0)2^5 + (0)2^6 + (1)2^7$$

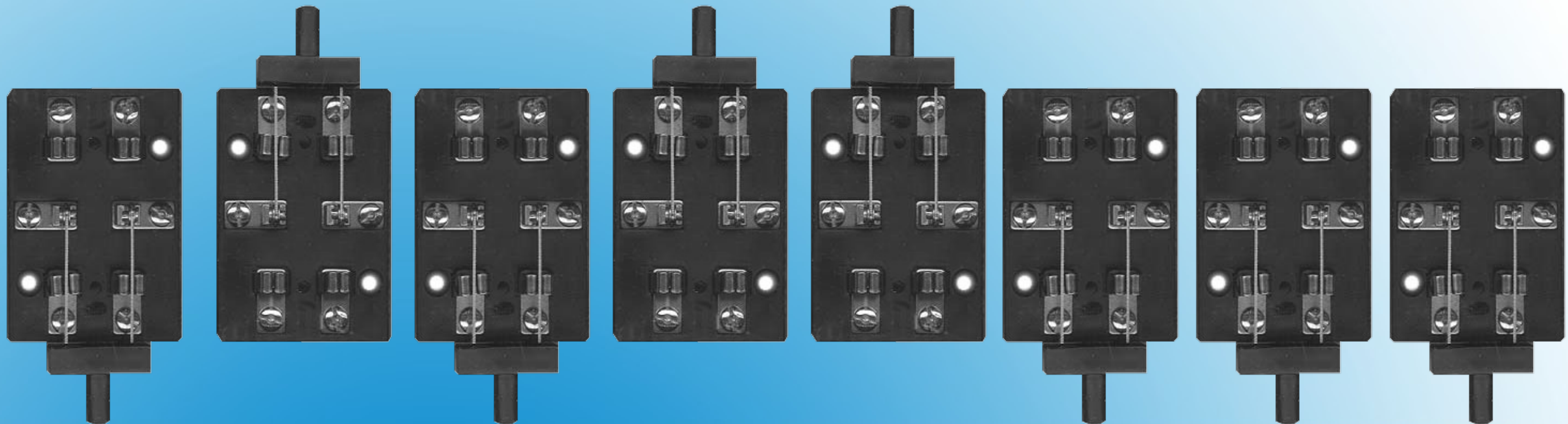
- There are other bases... hexadecimal (base 16), octal (base 8), and roman numerals or tally marks are an example of unary (base 1)

Binary

- What is this number?

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

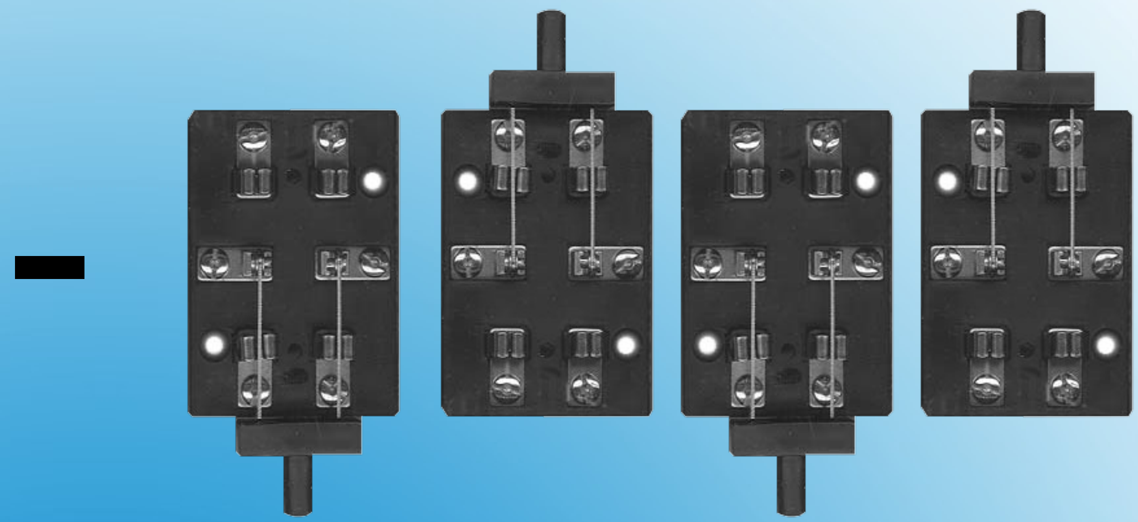
0 1 0 1 1 0 0 0

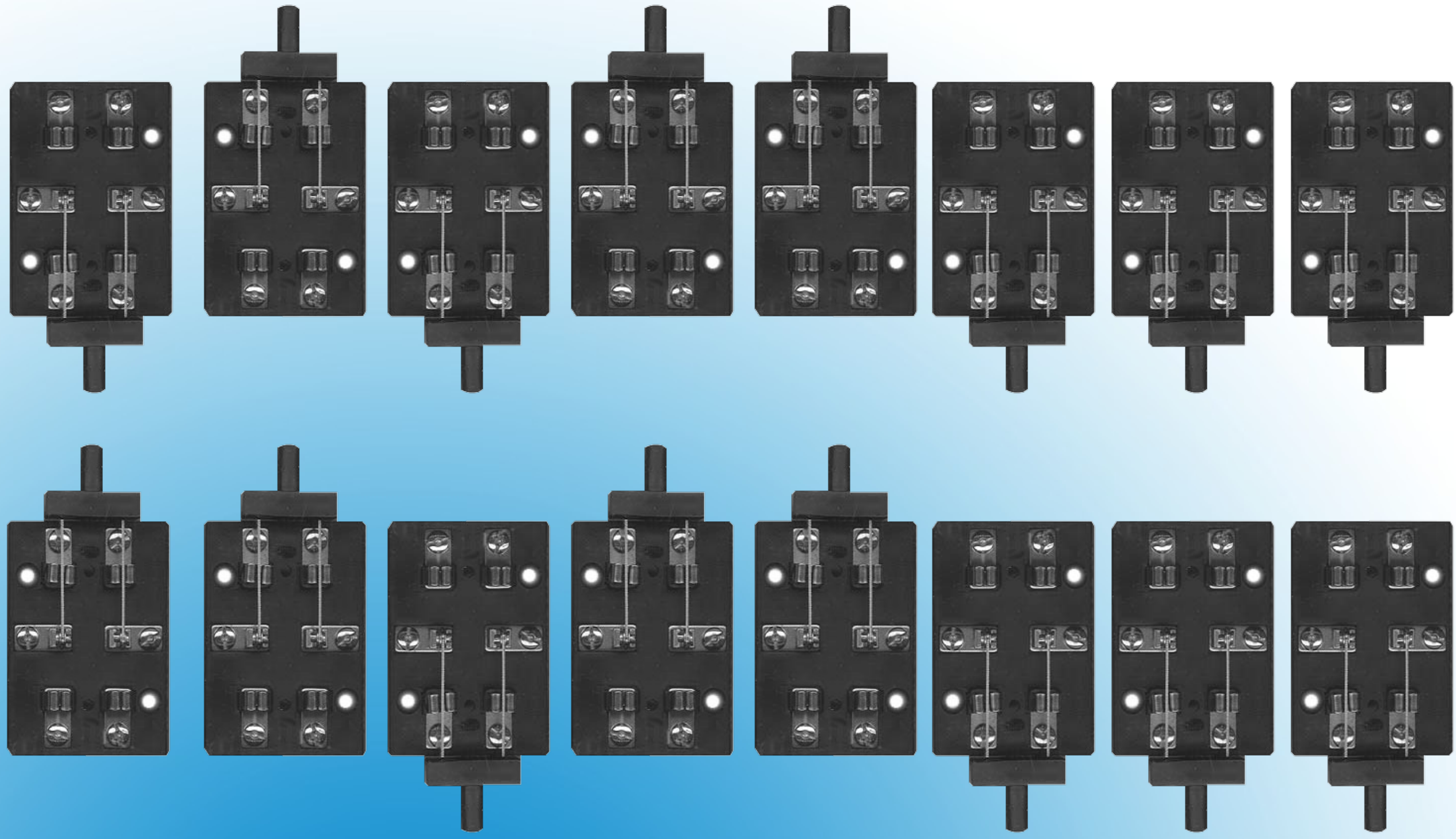


Binary

What About Negatives?

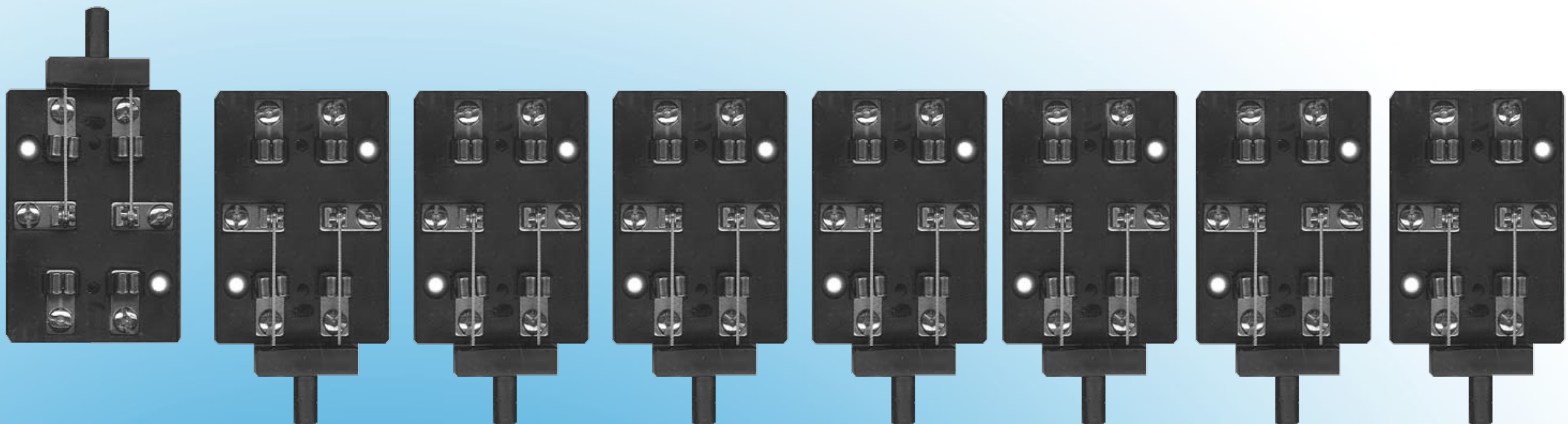
- How can we represent a negative number?





One's Complement

- Wait. What's this?



- Something called *two's complement* fixes this issue... but we won't get into that.

Weird!

- How can we representing letters using these switches?
- Surely there are examples of something like this?

1001101001000111

Representing Letters

- We can represent letters using a binary signal.
  - Pulse on versus a pulse off
- For instance: **Morse code**
  - Translating the pulses and their lengths into binary is easy
  - Dot: 1
  - Dash: 111
  - Gap between codes: 0
  - Gap between letters: 000
  - Gap between words: 0000000

Morse Code?

N I

11101000101

Morse Code to Binary

- Although, more trivially, we just represent characters as numbers.

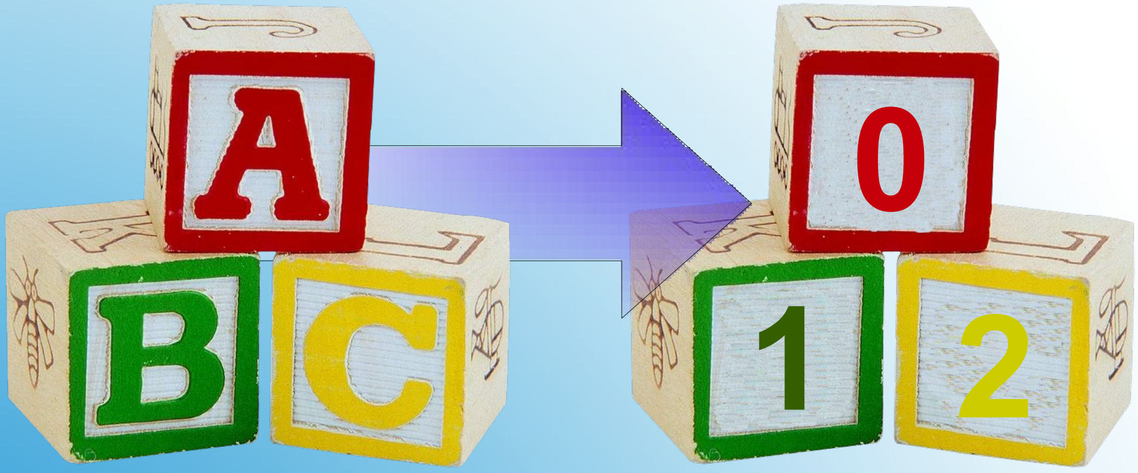
A = 0 = 00000

B = 1 = 00001

...

Y = 24 = 11000

Z = 25 = 11001



Character Mapping

- **String** a bunch of these sequences together...

11001011110011 10

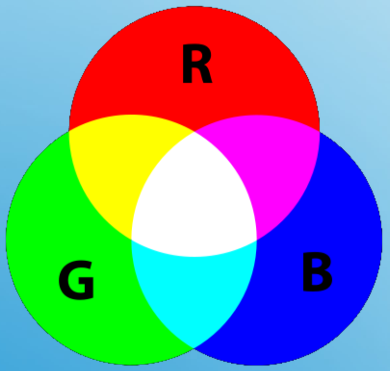
ZOO

- Such a scheme is typically used today:
  - The scheme is called ASCII (*American Standard Code for Information Interchange*)

Strings

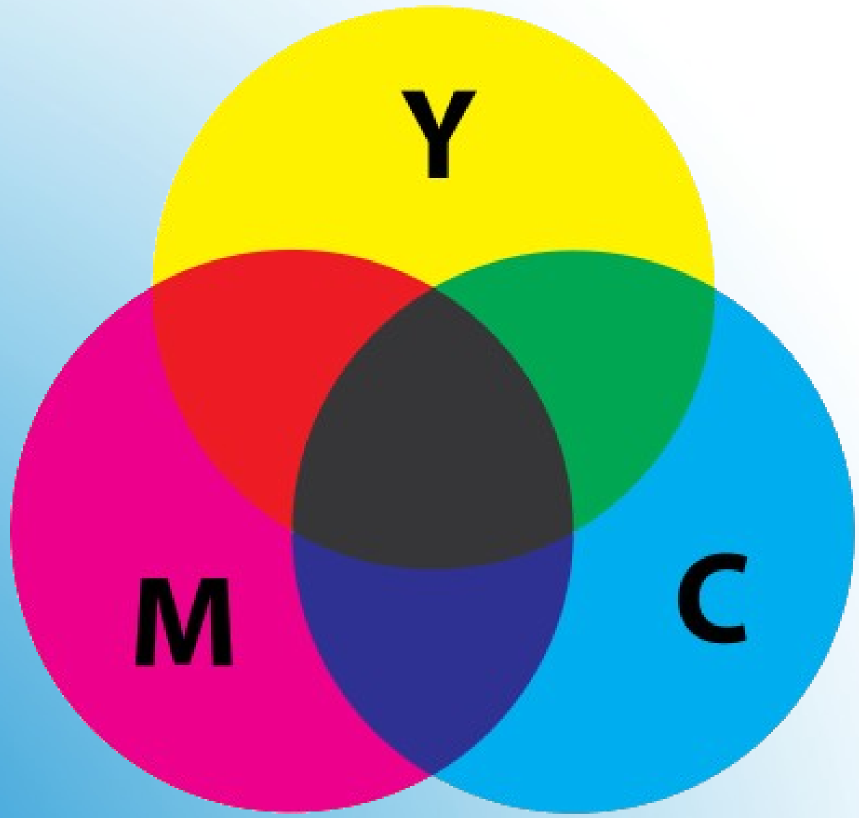
ASCII... art?

- Representing color is a tricky concept.
  - It is not necessarily finite (so we approximate)
  - How do we mix colors?



Representing Colors

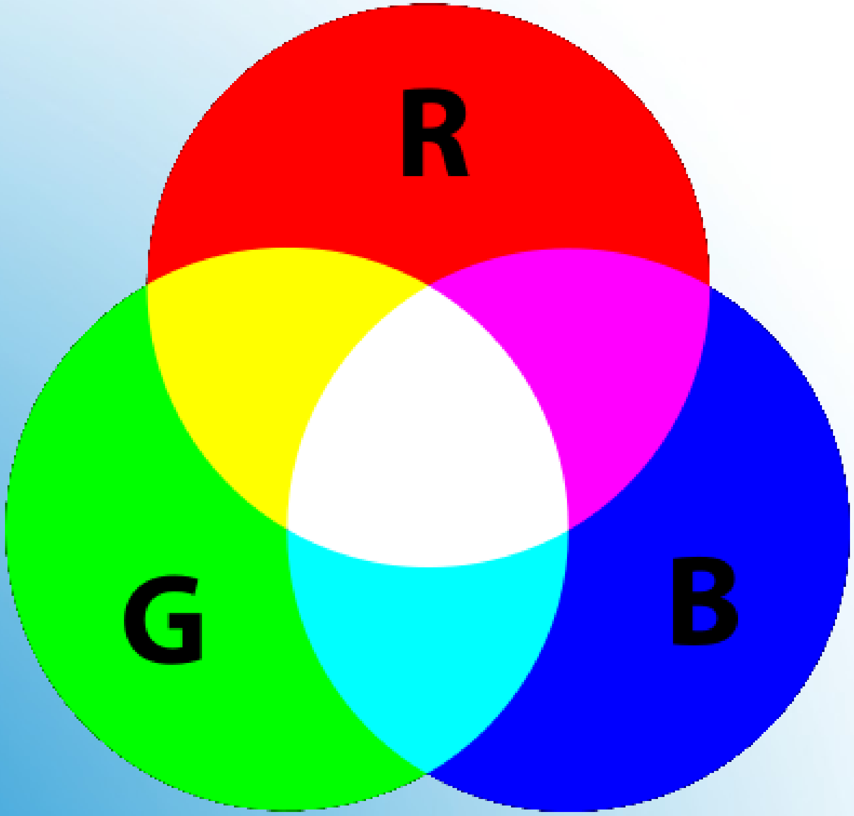
- This is what we commonly learn first. It represents the mixing of colors in dyes.
  - Used by printers
  - Normally, you'd have red, yellow, and blue as primary colors



Subtractive Color

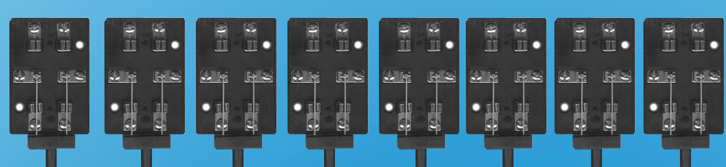
Additive Color

- This represents how light combines to form different colors.
  - Typically how CRT monitors work is that they shoot red, green, and blue beams.
  - These combine to form the *pixels* you see.

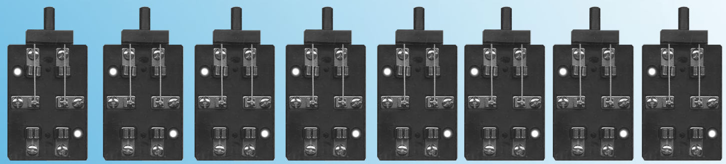


- We know that 8 switches (a *byte*) can hold 256 values.
- Studies have shown that humans cannot differentiate between around 200 shades.
- Therefore, each “dye” is represented by a byte.

No red



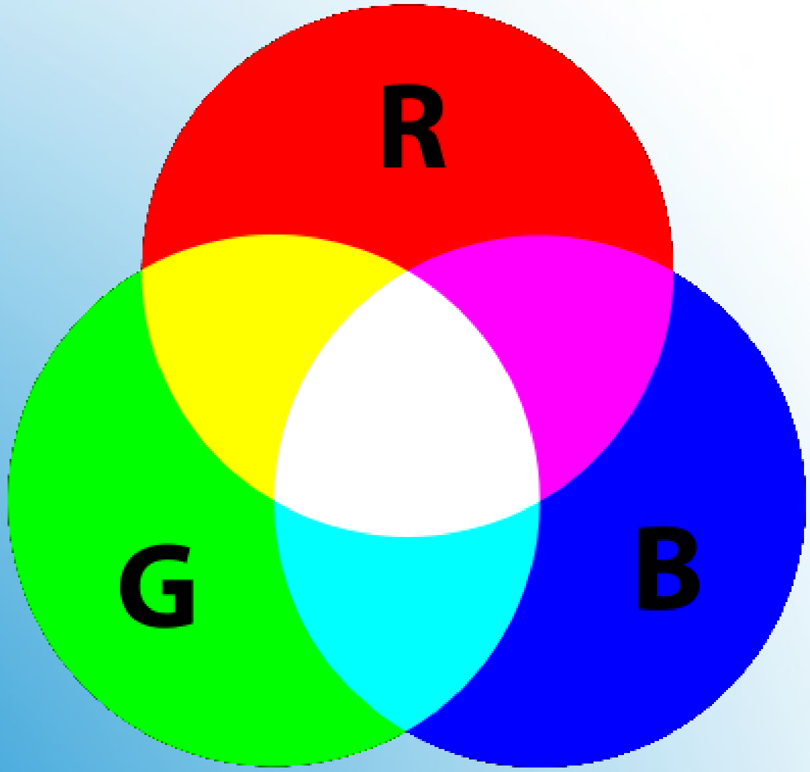
Lots of red



Approximation

Representing Color

- We just need the *primary colors*.
- So three values (three bytes) will suffice for any color!
- What would
  00000000 red
  11111111 green
  11111111 blue
  yield?



- An image is just a collection of small dots.
- A *pixel* (picture element) is the smallest such dot.
- A pixel is just a color.

pixel a == r:11111111 g:00000000 b:11111111

pixel b == r:11111111 g:11111111 b:00000000

pixel c == r:11111111 g:00000000 b:00000000

Pixels

Representing Images

- An image is just a collection of pixels!

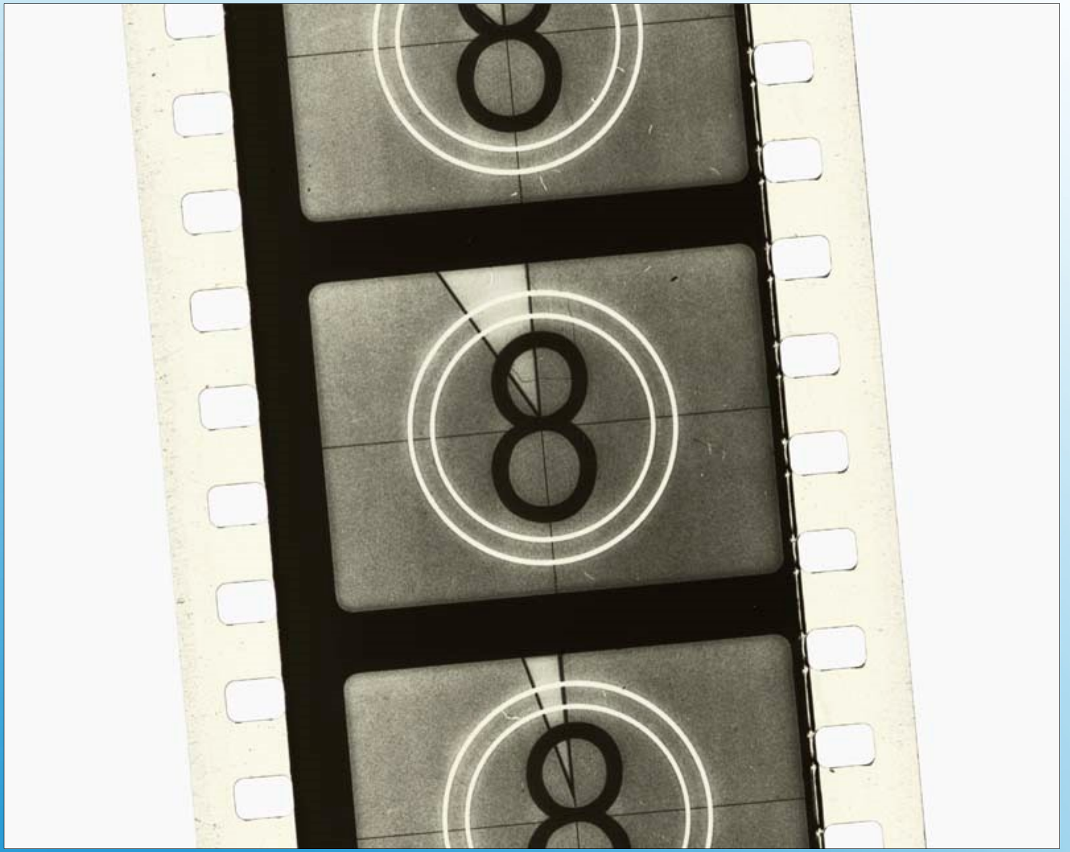
Red:	175	==	10101111
Green:	150	==	10010110
Blue:	109	==	01101101

- We can now encode the shrubbery for the **Knights that say 11101000101**
  - Or any image our heart so desired...

10011101001000111
0100100111011011
1000101110001001
0110110011101101
0110100100101001
0110100010001010

_It_ is a good shrubbery!

- And videos are simply collections of images!



Representing Videos

- Alright... convinced you can do everything with switches?
  - Good.
  - Alright, fine. Maybe we will get to *sound* later in the course. (It's a lot trickier)
- Let us now *abstract all information as a set of switches*.
  - What can we do with these switches?
  - Add? Subtract? Multiply?
  - How is the possible?

Mathematical Operations

- The answer: The same way we always do it.
  - Simply add the digits
  - If greater than or equal to the base (10) then carry a 1!

$$\begin{array}{r} 2363 \\ +\ 1289 \\ \hline 3652 \end{array}$$

Decimal Addition

- The answer: ***The same way we always do it.***
  - Simply add the digits
  - If greater than or equal to the base (10) then carry a 1!

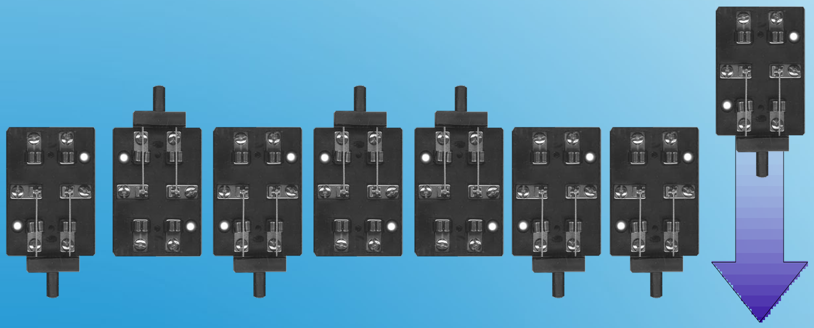
$$
\begin{array}{r}
0011 \\
\underline{+\ 0101} \\
1000
\end{array}
$$

Binary Addition

- Multiplication can be expressed as addition:

$$2 * 5 == 2 + 2 + 2 + 2 + 2 == 5 + 5$$

- Of course, more complicated expressions benefit from a more sophisticated approach:
  - What happens if I shift bits over one?



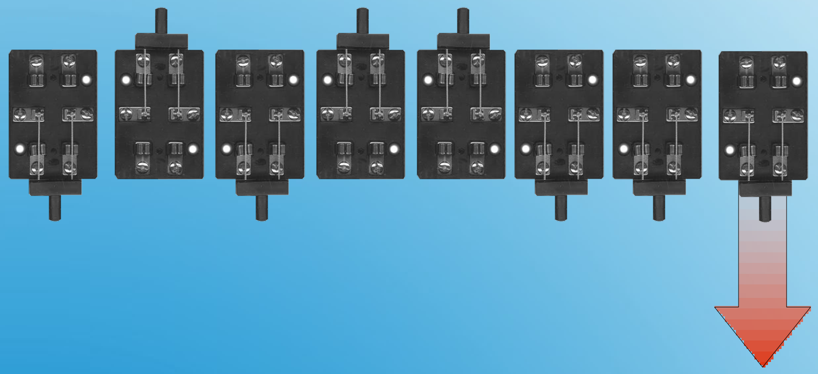
Binary Multiplication

- If we factor a larger multiplication into a series of multiplications of 2...
  - we can use simple shifts and additions to multiply!
  - A technique in common use is **Booth's Algorithm**

Binary Multiplication

- Division is *roughly* the inverse of multiplication.

- Knowing what we know about multiplication, how can we divide?

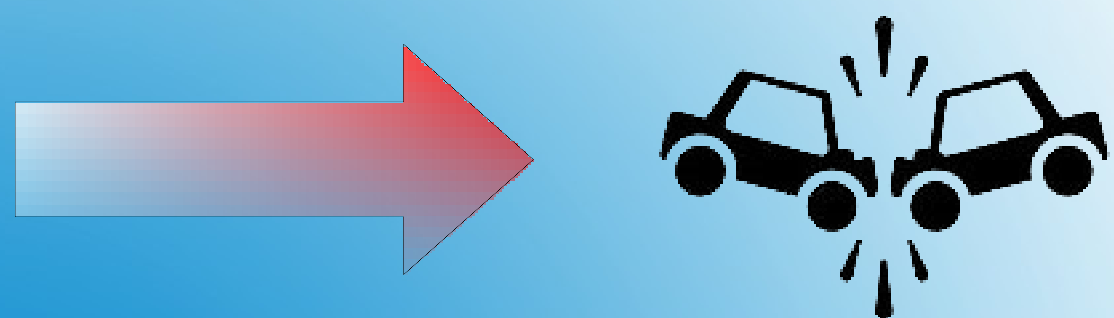


Binary Division

- Of course, there are also techniques and algorithms related to efficient division.

- Take note that not all things are divisible on a machine:
  - Most computers do not respect division by zero!

x / 0



Binary Division

- Computers use a simple architecture to perform tasks.
- Computers represent numbers using switches called **bits**.
- This is a **binary** representation, and it is sufficient for any *finite data*.
  - Or anything that can be approximated as finite
  - Colors, images, sound, video...
- This representation can be manipulated mathematically just like decimal numbers!

What have we shown?

- We will now learn a programming language.
  - Languages are often inspired by either the theoretical or physical nature of the machine.
  - Variables might be constrained to a certain number of switches.
  - Ask yourself **why** certain design decisions of the language were made.

Keep in Mind