Memory

Recap and Agenda

Last week we looked at

- Integrated circuits on our breadboards.
- Building an adder, transforming our breadboards into little binary calculators!

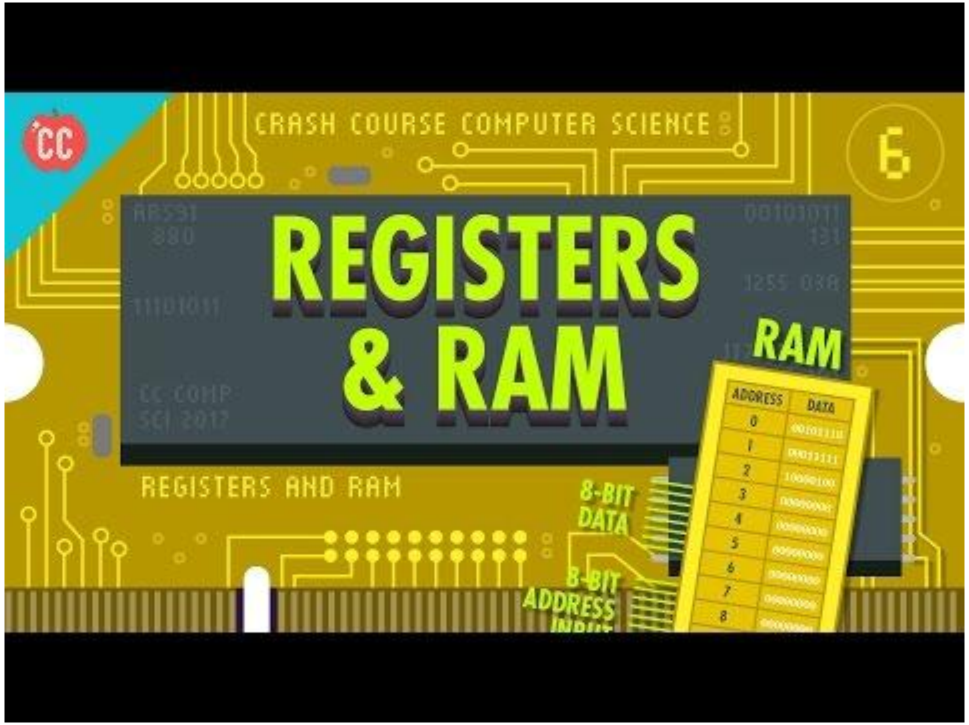
Today, we'll see:

- How computers store information, using *latches*.
- How memory works more broadly.

Our ALU is lonely

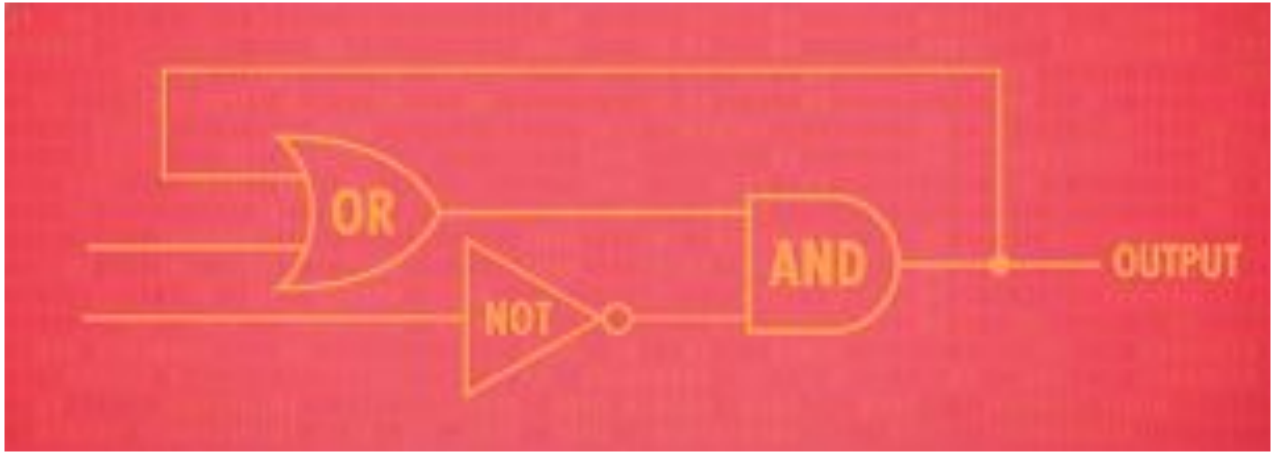
- The ALU on its own it’s very interesting because the results of our calculations aren’t *stored* anywhere. But how can a computer store information at all?
- The trick is to “trap” the flow of electricity by creating feedback loop within a circuit. The presence of this “trapped electricity” represents a binary digit 1, whereas its absence represents a 0.
- By combining many of these circuits, we can store larger quantities of data.

Details? Take it away, Carrie-Anne!



Constructing our very own AND-OR latch

Using the logic gate ICs you're now all very familiar with, try to build an AND-OR latch. Here's the diagram from the video:



Scaling up

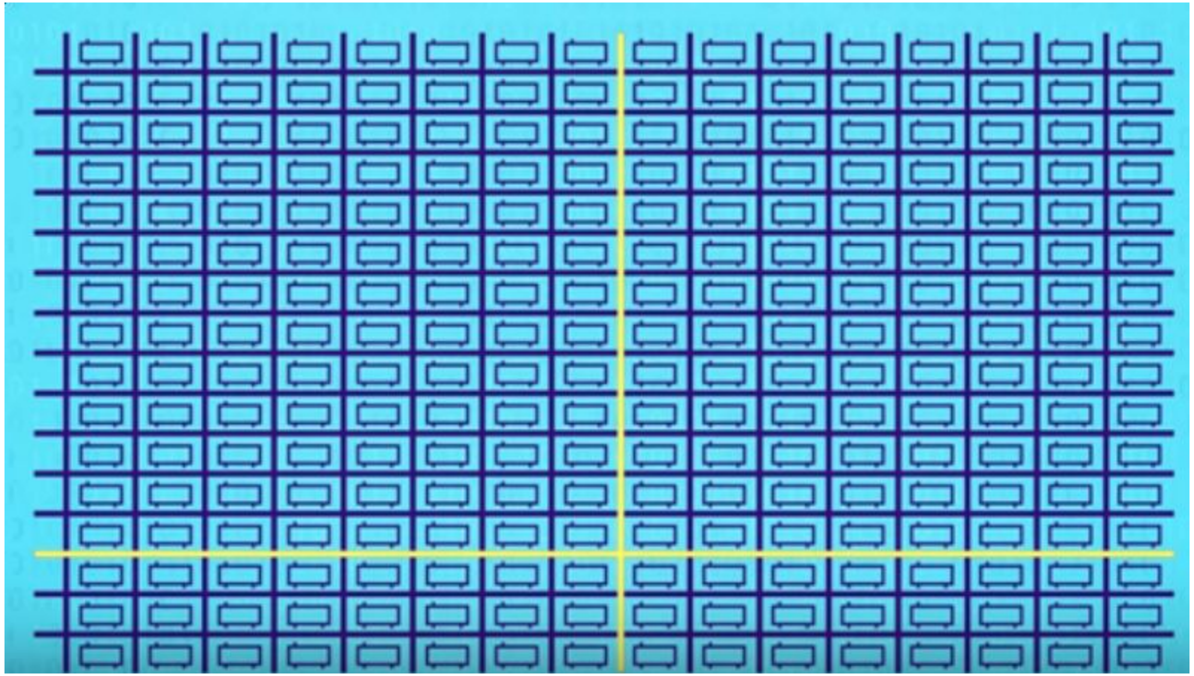
- By putting together many 1-bit memory units like the one you just built, we can construct larger memory units.
- By joining eight 1-bit units, we can store one *byte*.
- Finally, by using many 1-byte units, we can construct our computer’s *main memory*, or *RAM.*
- We also need memory units called *registers* which are the CPU’s internal memory. The CPU uses the contents of registers for operations in the ALU.

Where are you??

- A CPU is capable of *reading* data from RAM into its registers, and *writing* data from registers into RAM. But how do we specify *where* to read from and write to?
- The idea is to use **addresses**, which are just numbers referring to a particular 1-byte memory unit.
- If we have 256 1-byte cells in our RAM, then we need 8 bits to represent an address.

But how do addresses work?

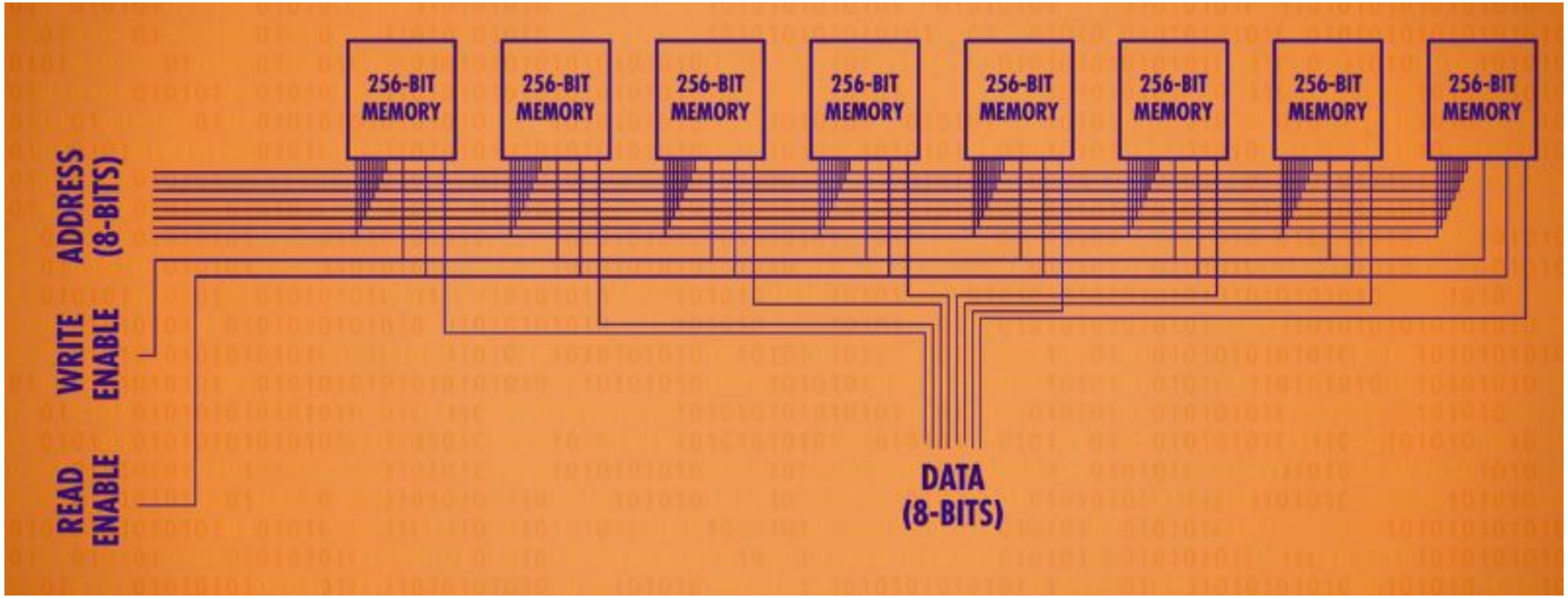
- The idea is to arrange 1-bit memory units into a **matrix**.
- To select a particular bit, it suffices to specify the *row* and *column* number of the bit.
- In a 16x16 grid, we need 4 bits for the row, and 4 for the column



What's the address of the selected 1-bit memory cell?

But that’s just 256 *bits*, not 256 *bytes*!

- So what? Just put eight 256-bit memory units side by side, and now we can store 256 bytes.



It’s just a big list of bytes

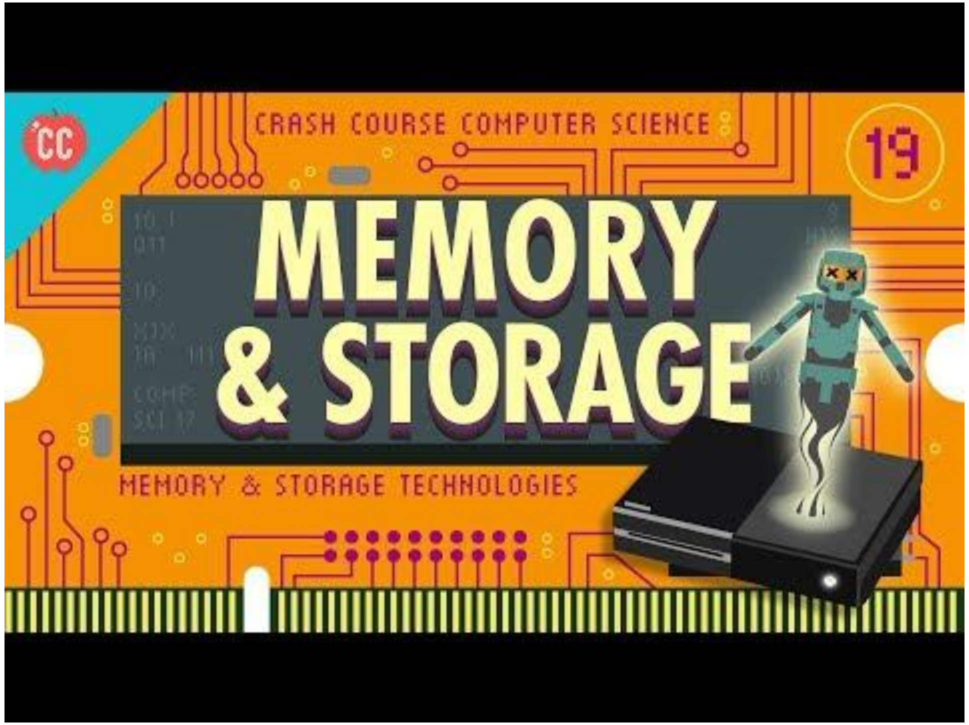
- And there’s our computer’s RAM unit, right there!
- We can specify an address, which will activate the corresponding cell in each of the 256-bit memory units.
- We can set the RAM unit into reading mode or writing mode, using dedicated wires.
- In writing mode, the specified address will be set to the specified 8-bit value on the data wires.
- In reading mode, the 8-bit value contained at the specified address will be output onto the data wires.

RAM isn’t the only place we store data

- RAM, due to the way it’s designed, requires a constant flow of electricity in order to continue to store data. If we pull the plug on a computer, the RAM is flushed.
- We need need a strategy for storing data that doesn’t rely on constant electrical current.

Magnets! How do they work anyway?

Take it away Carrie-Anne!



Recap

This week we saw:

- How to build our own 1-bit memory unit using logic gates in a feedback loop
- How we can scale up our tiny memory units into larger blocks to form a computer’s main memory, or RAM.
- How computers use different types of memory for different purposes: hard disks for storage, RAM for storing programs and temporary data, and registers as go-betweens for communicating with the ALU.

Next week we will:

- Design our own fantasy processor.
- Write and execute programs in machine code.