Programs, the CPU, and Memory

Before we get our hands dirty with learning how to build a computer, let's talk theory first. In an earlier lesson, we talked about binary and how computers perform calculations. Remember that our computer can only communicate in binary using 1's and 0's. Our computers speak in machine language but we of course speak in human languages like English, Spanish, Mandarin, Hindi, you get the idea. If we want to communicate with our machines we have to have some sort of translation dictionary, just like if I wanted to say something in spanish I'd look it up in an English to Spanish dictionary. Well our computers have a built in translation book. In this lesson we'll dive deeper into how our computer translates the information we give it into instructions that it understands. RIght now you're probably using a web browser, music player, text editor or something else on your computer. We interact with these applications on a daily basis, they're referred to as programs. Programs are basically instructions that tell the computer what to do. We typically store programs on durable media like hard drives, you can think of programs like cooking recipes, we keep these recipes all stored together in a cookbook, just like apps stored in a hard drive. Now we want to make a ton of food, so we hire a chef to follow our recipes and whip up something good. The faster our chef works, the more food she'll prepare, the chef is our CPU she processes the recipes, we send her and makes the food, our chef works super fast so fast that she can cook faster than she can read. So we take copy of the recipes and put them into RAM. Remember that RAM is our computers short term memory, it stores information in a location our CPU can access it faster than they could with our hard drive. Now we can give our chef one or two recipes at a time instead of reciting the entire cookbook to her. Okay, now let's say I want to make a peanut butter and jelly sandwich. I see a pretty good recipe and send it to our chef to make, remember that our chef needs these instructions quickly so I don't send her the entire recipe, I sent her one line at a time. One, get two slices of bread, two, put peanut butter on one slice, three, put jelly on another slice, four, combine the two slices of bread. Now let me throw one more thing at you. Our chef can only communicate with us in 1's and 0's. So instead of sending something readable, like the recipe for peanut butter and jelly sandwich, we have to center something like this. In reality this process is a little more complicated. Our CPU is constantly taking instructions and executing them, these instructions are written in binary. But how do they travel around the computer? In our computer, we have something called the external data bus or EDB. It's nothing like a bus at all, it's a row of wires that interconnect the parts of our computer, kind of like the veins in our body. When you send a voltage to one of the wires, we say the state of the wire is on are represented by a 1, if there's no voltage then we say that the state is off represented by a 0. This is how we send around our 1's and 0's, sound familiar? In the last lesson, we talked about how transistors help us to send voltages. Now we know how our bits physically travel around the computer, the EDB comes in different sizes, a bit, 16-bit, 32, even 64. Can you imagine if you had 64 wires going you can move around a lot more data right now we're just going to stick with using an EDB with 8-bits in our examples, sending one byte at a time. Okay, so now our CPU is receiving a byte and it needs to get to work. Inside the CPU, there are components known as registers, they

let us store the data that our CPU works with. If for example our CPU wanted to add two numbers, one number would be stored in a register A, another number will be stored in register B, the result of those two numbers will be stored in register C. Imagine the register is one of our chefs work tables, since she has a place to work, she can start to cook, to do so she uses a translation book to translate her binary into tasks that she can perform. Let's jump back for a second. Remember that our programs are copied into RAM for the CPU to read, RAM is memory that's randomly accessed allowing our CPU to read from any part of RAM as quickly as any other part. We don't actually send data from RAM over the EDB, there would be way too much stuff. RAM can hold millions even billions of rows of data. Despite our sandwich example, most of our recipes aren't simple at all. There can be thousands of lines long, we want to process them and we don't actually go in any particular order. Since we can only send one line of data through the EDB at a time we need the help of another component, the memory controller chip or MCC.

The MCC is a bridge between the CPU and the RAM. You can think of it like a nerve in your brain connecting to your memories, the CPU talks to the MCC and says hey I need the instructions for step number three of this recipe, the MCC finds instructions for step number three in RAM, grabs the data and sends it through the EDB. There's another bus that's nothing like a bus involved in the process called the address bus, it connects the CPU to the MCC and sends over the location of the data but not the data itself, then the MCC takes the address and looks for the data and then data is then sent over the EDB. Believe it or not, RAM isn't the fastest way we can get more data to our CPU for processing. The CPU also uses something known as cache.

Cache is smaller than RAM but it lets us store data that we use often and let's just quickly reference it. Think of RAM like a refrigerator full of food, it's easy to get into but it takes time to get something out, on the flip side of that, cache is like the stuff we have in our pockets, it's used to store recently or frequently accessed data. There are three different cash levels in a CPU, L1, L2 and L3. L1 is the smallest and fastest cache. So now we understand how our RAM interacts with our CPU but how does our CPU know when a set of instructions end and a new one begins? Our CPU has an internal clock that keeps its operations in sync. It connects to a special wire called a clock wire. When you send or receive data, it sends a voltage to that clock wire to let the CPU know it can start doing calculations. Think of our clock wires as the ticking of a clock, for every tick the CPU does one cycle of operations. When you send a voltage to the clock wire as referred to as a clock cycle, if you have lots of data you need to process in the command you need to run lots of clock cycles. Have you ever seen a CPU in the store and has something labeled 3.4 Ghz? This number refers to the clock speed of the CPU, which is the maximum number of clock cycles that it can handle in a certain time period. 3.40 GHz is 3.4 billion cycles per second, that's super fast. But just because it can run at this speed doesn't mean it does, it just means that it can't exceed this number, still, that number doesn't stop some people from trying. There's a way you can exceed the number of clock cycles on your CPU on almost any device, it's referred to as over clocking and it increases the rate of your CPU clock cycles in order to perform more tasks. This is commonly used to increase the performance in low end CPUs, let's say you're a gamer and you want to have better graphics and less lag while playing, you might want to over clock your CPU when you play the game. But there are cons to doing this like potentially overheating your CPU.

Revision #6
Created 2023-05-29 09:02:17 UTC by naruzkurai
Updated 2023-07-05 14:09:22 UTC by naruzkurai