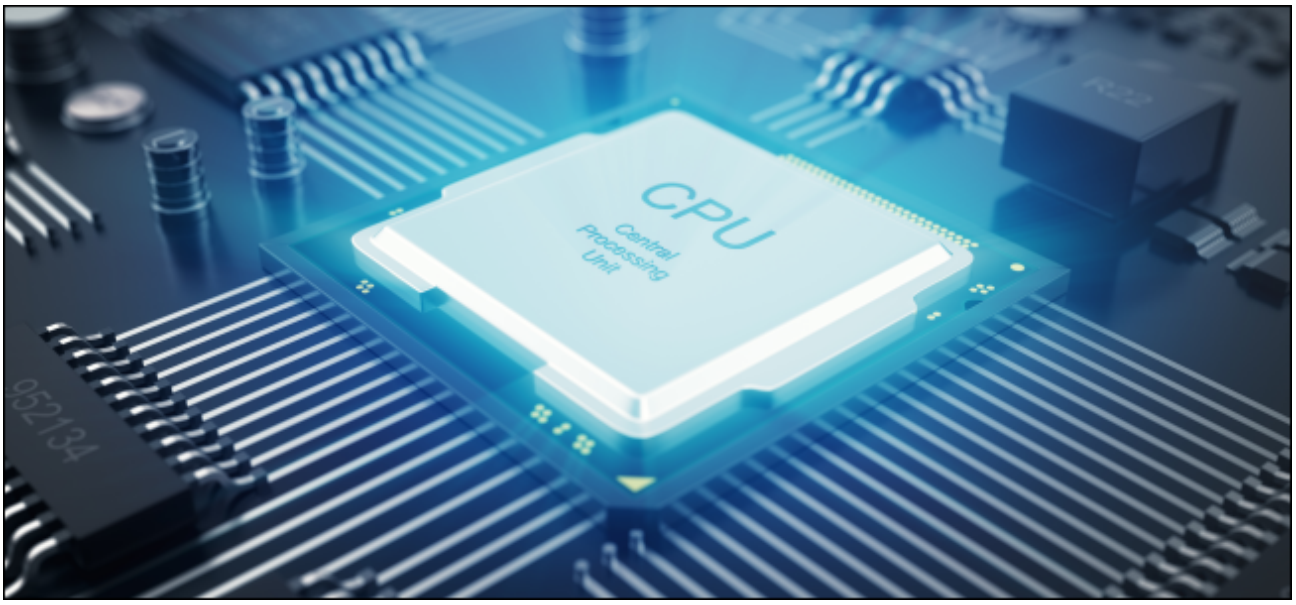
How Does a CPU Actually Work?

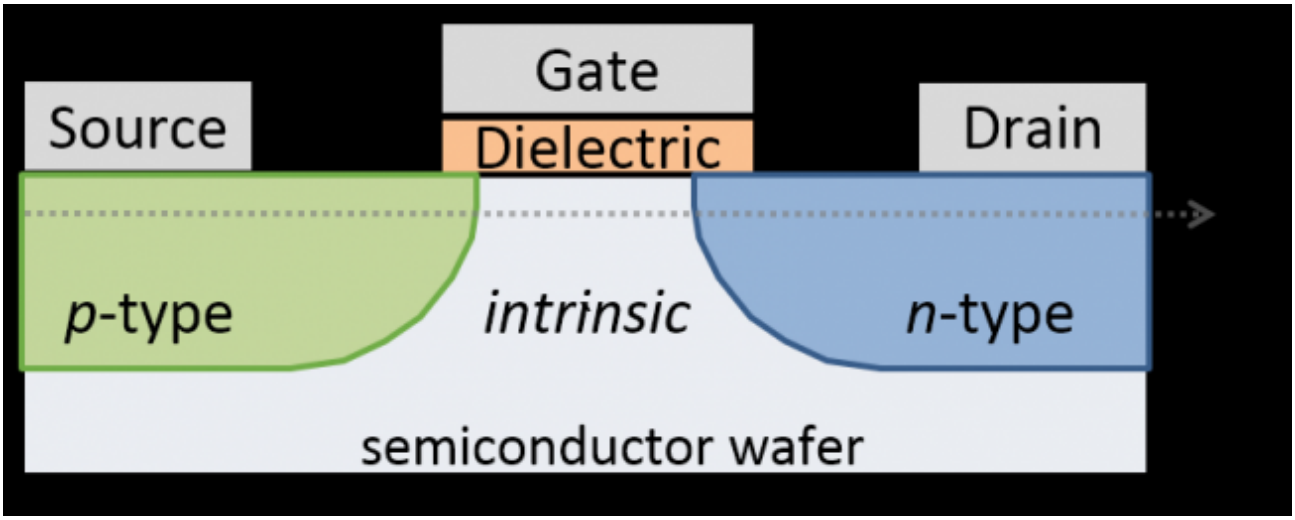


Most things in a computer are relatively simple to understand: the RAM, the storage, the peripherals, and the software all work together to make a computer function. But the heart of your system, the CPU, seems like magic even to many tech people. Here, we'll do our best to break it down.

Most of the research for this article comes from "But How Do It Know?"[1] by J. Clark Scott. It's a fantastic read, goes into much more depth than this article will, and is well worth the couple bucks on Amazon.

One note before we begin: modern CPUs are orders of magnitude more complex than what we're outlining here. It's nearly impossible for one person to understand every nuance of a chip with over a billion transistors. However, the basic principles of how it all fits together remain the same, and understanding the basics will give you a better understanding of modern systems.

Starting Small



Computers operate in binary[2]. They only understand two states: on and off. To perform calculations in binary, they use what's called a transistor. The transistor only allows the source current to flow through it to the drain if there is current across the gate. Essentially, this forms a binary switch, which cuts the wire off depending on a second input signal.

RELATED: ***What is Binary, and Why Do Computers Use It?***[3]

Modern computers use billions of transistors to perform calculations, but at the lowest levels, you only need a handful to form the most basic components, known as gates.

Logic Gates

Stack a few transistors properly, and you have what's known as a logic gate. Logic gates take two binary inputs, perform an operation on them, and return an output. The OR gate, for example, returns true if either of the inputs is true. The AND gate checks if both inputs are true, XOR checks if only one of the inputs are true, and the N-variants (NOR, NAND, and XNOR) are inverted versions of their base gates.

Doing Math With Gates

With just two gates you can do basic binary addition. This diagram above shows a half adder, created using Logicly[4], a free online playground for logic gates. The XOR gate here will turn on if just one of the inputs is on, but not both. The AND gate will turn on if both inputs are on, but

stay off if there's no input. So if both are on, the XOR stays off, and the AND gate turns on, coming to the correct answer of two:

This gives us a simple setup with three distinct outputs: zero, one, and two. But one bit can't store anything higher than 1, and this machine isn't too useful as it only solves one of the simplest math problems possible. But this is only a half adder, and if you connect two of them with another input, you get a full adder:

The full adder has three inputs—the two numbers to add, and a "carry." The carry is used when the final number exceeds what can be stored in a single bit. Full adders will be linked in a chain, and the carry is passed from one adder to the next. The carry is added to the result of the XOR gate in the first half adder, and there's an extra OR gate to handle both cases when the so that would need to be on.

When both inputs are on, the carry turns on, and sends it to the next full adder in the chain:

And this is about as complex as addition gets. Moving up to more bits essentially just means more full adders in a longer chain.

Most other math operations can be done with addition; multiplication is just repeated addition, subtraction can be done with some fancy bit inversion, and division is just repeated subtraction. And while all modern computers have hardware-based solutions to speed up more complicated operations, you can technically do it all with the full adder.

The Bus, and Memory

Right now, our computer is nothing more than a bad calculator. This is because it can't remember anything, and does nothing with its outputs. Shown above is a memory cell, which can do all of that. Under the hood, it uses a lot of NAND gates, and in real life can be quite different depending on the storage technique, but its function is the same. You give it some inputs, turn on the 'write' bit, and it will store the inputs inside the cell. This isn't just a memory cell, as we also need a way to read information from it. This is done with an enabler, which is a collection of AND gates for each bit in the memory, all tied to another input, the "read" bit. The write and read bits are often called "set" and "enable" as well.

This whole package is wrapped up into what's known as a register. These registers are connected to the bus, which is a bundle of wires running around the whole system, connected to every component. Even modern computers have a bus, though they may have multiple buses to improve multitasking performance.

Each register still has a write and read bit, but in this setup, the input and output are the same thing. This is actually good. For example. If you wanted to copy the contents of R1 into R2, you would turn on the read bit for R1, which would push the contents of R1 onto the bus. While the read bit is on, you'd turn on the write bit for R2, which would copy the bus contents into R2.

Registers are used to make RAM as well. RAM is often laid out in a grid, with wires going in two directions:

The decoders take a binary input and turn on the corresponding numbered wire. For example, "11" is 3 in binary, the highest 2-bit number, so the decoder would turn on the highest wire. At each intersection, there's a register. All of these are connected to the central bus, and to a central write and read input. Both the read and the write input will only turn on if the two wires crossing over the register are also on, effectively allowing you to select the register from which to write and read. Again, modern RAM is far more complicated, but this setup still works.

The Clock, the Stepper, and the Decoder

Registers are used everywhere and are the basic tool for moving data around and storing information in the CPU. So what tells them to move things around?

The clock is the first component in the core of the CPU and will turn off and on at a set interval, measured in hertz, or cycles per second. This is the speed you see advertised alongside CPUs; a 5 GHz chip can perform 5 billion cycles per second. Clock speed is often a very good metric for how fast a CPU is.

The clock has three different states: the base clock, the enable clock, and the set clock. The base clock will be on for half a cycle, and off for the other half. The enable clock is used to turn on registers and will need to be on for longer to make sure that the data is enabled. The set clock always needs to be on at the same time as the enable clock, or else incorrect data could be written.

The clock is connected to the stepper, which will count from one to the max step, and reset itself back to one when it's done. The clock is also connected to AND gates for each register that the CPU can write to:

These AND gates are also connected to the output of another component, the instruction decoder. The instruction decoder takes an instruction like "SET R2 TO R1" and decodes it into something that the CPU can understand. It has its own internal register, called the "Instruction Register," which is where the current operation is stored. How exactly it does this comes down to the system you're running on, but once it's decoded, it will turn on the correct set and enable bits for the correct registers, which will fire off in accordance to the clock.

Program instructions are stored in RAM (or L1 cache on modern systems, closer to the CPU). Since program data is stored in registers, just like every other variable, it can be manipulated on the fly to jump around the program. This is how programs get their structure, with loops and if statements. A jump instruction sets the current location in memory that the instruction decoder is reading from to a different location.

How It All Comes Together

Now, our gross oversimplification of how a CPU works is complete. The main bus spans the whole system and connects to all of the registers. The full adders, along with a bunch of other operations, are packed into the Arithmetic Logic Unit, or the ALU. This ALU will have connections to the bus, and will also have its own registers for storing the second number it's operating on.

To perform a calculation, program data is loaded from system RAM into the control section. The control section reads two numbers from RAM, loads the first one into the ALU's instruction register, and then loads the second one onto the bus. Meanwhile, it sends the ALU an instruction code telling it what to do. The ALU then performs all the calculations and stores the result in a different register, which the CPU can read from and then continue the process.

Image Credit: Rost9[5]/Shutterstock

Links

1. https://www.amazon.com/But-How-Know-Principles-Computers-ebook/dp/B00F25LEVC?tag=823814-20
2. https://www.howtogeek.com/367621/what-is-binary-and-why-do-computers-use-it/
3. https://www.howtogeek.com/367621/what-is-binary-and-why-do-computers-use-it/
4. https://logic.ly/demo/samples
5. https://www.shutterstock.com/image-illustration/circuit-board-technology-background-central-computer-746807584?src=gNjsawPqjdAQO3mEcljNrQ-1-2