Computer Guts

Most people are unaware when they open a tab, launch an app, or send an email to their friends, what the computer they're using is doing behind the scenes. This essay is not about what happens behind the scenes. It is concerned with what happens behind *that*, all the way to the 1s and 0s, that are so vital to our modern world. These efficient low-level systems are responsible for the smooth and speedy running of our computers.

The first step in this ladder is the operations that can be done on the 1s and 0s. Computers use a special number system that uses only two digits, 1 and 0, called the binary system. One binary digit is called a bit and a group of 8 of them is called a byte. Every decimal number can be represented in binary and vice versa. Take the number 598 in decimal. If you multiply each of the digits by 10 to the power of the place they're in and add them together, you get that same number. $5x10^2 + 9x10^1 + 8x10^0 = 500 + 90 + 8 = 598$. The same follows for binary; if you take the number 110_2 it equals $1x2^2 + 1x2^1 + 1x2^0 = 4 + 2 + 0 = 6$. It's helpful to remember what the powers of 2 are: 2, 4, 8, 16, and that the rest is just the previous doubled. There are some basic operations you can do on binary numbers. These logic gates are so basic that they have no counterpart in the decimal system. The AND gate will output true if both of the inputs are true. For instance, if you have bread AND ham, you can make a dull sandwich. If you don't have one of them - the bread, or the ham - or neither, then you can’t make a sandwich. The OR gate outputs true if one of the inputs is true. If you win a bet OR get a good grade, you feel happy. The simplest of the basic logic gates is the NOT gate. What comes in, the opposite comes out. By combining these three gates in different configurations, you can make more complex ones.

The next level in this hierarchy is the part of the computer that will do the computing. That is the task of the ALU[1], to quickly compute addition, subtraction, and some logic operations. By combining a few of the simple logic gates described earlier, more complex ones can be made. By combining a few AND, OR, and NOT gates, a half-adder can be created. The half-adder can't, however, add any multi-digit numbers. A full adder is composed of two half adders, creating a gate with three inputs and two outputs (think about carrying over). By combining 16 of these in series, you can create a 16-bit adder, which can add 16-bit numbers, taking care of addition. One way to do subtraction is called the two's complement. It treats subtraction as a special case of addition, except that one of the numbers is negative, similar to how the answer to 5 - 2 is the same as 5 + (-2). There are two ways to implement it, the easier one is to negate the number you want to subtract and add it to the other number. There are some similarities between these operations. The only difference between addition and subtraction is that in subtraction, you negate one of the numbers first. The difference between OR and AND is that either everything is negated, or it's not. Because of this, you can control what operation is done, simply by negating the inputs accordingly. The one drawback is that the ALU, by itself, can't do much. Whatever you compute gets computed, but not saved. This is done by the RAM.

The RAM[2] is the short-term memory of the computer. One way to visualize it is to imagine it as a long, numbered list. It's unsuitable for storing, say files but excellent for arbitrary numbers that you might need in a moment. When inputting an address in the RAM, you should get a 1-byte number from that address. There are just two primitive gates that the whole thing is based on. A 1-bit register can remember and output the one-bit it remembers. Combining it with another gate, the multiplexer, it becomes much more useful. The multiplexer selects from its inputs what to pass onto the output. What tells the multiplexer what to select is usually the

[1] The Arithmetic and Logic Unit
[2] The Random Access Memory

address of what you want RAM to recall. The result of combining them is a 1-byte register, which can store 8 bits. After mashing together multiple 1-byte registers, you end up with a 16-byte register. 64-byte memory can be made of four 16-byte registers. On and on, this hierarchy is continued, until the address required to select a specific byte is 16 bits long. The RAM can be used to store many things, numbers representing numbers, numbers representing letters, numbers representing pictures, and much more. However, some things are impractical for the RAM to store, like programs. It's the ROM that keeps those.

The ROM[3] is the program memory, but it usually only stores the computer's operating system. The operating system makes the computer more user-friendly, for example, the Windows OS creates the icons for all the applications on the computer. The ROM's very similar to the RAM in almost every respect, except that once set, the ROM can not change its content (hence the name). The information kept in the ROM chip is permanently stored, even after the computer is turned off. The difference between the RAM and the ROM is very important for the assembly compiler.

The assembly compiler is the final stage in the hardware hierarchy of the computer. When someone opens an application the app's program is put into something called a compiler. The compiler takes the program and converts it into a lower-level version of the program. Then the computer takes that new program and translates it into another, even simpler language. At some point, the commands that we feed into our computers are turned into operations that the ALU and RAM work with, the calculator portion of the computer. That last conversion is the job of the assembly compiler. It converts a program from assembly to machine language, a messy unreadable jargon that computers read. The machine language is plugged into the ALU and RAM and they evaluate it. The assembly language has access to the RAM to store information short-term, but the connection to the RAM takes some time to use. Remember that there are many, many systems and programs based on these components. If a computation is a billionth

[3]The Read Only Memory

of a second slower, that will noticeably impact the speed of the computer. This journey to the RAM is replaced with something called a register, which also stores information, but in much smaller quantities. An assembly compiler can be used to create many things, simple applications, more complex applications, and even other programming languages. Using those higher-level languages some very complex software can be designed. Most operating systems, for example, are written in C, a programming language created with assembly.

The computer can be visualized as a hierarchy, at the bottom you have the 1s and 0s and logic gates, and on top of that, the ALU is defined. Separately from the ALU, the RAM and ROM store short- and long-term memory. The assembler runs programs while referencing each of those lower-level systems. The more you know about hardware and computers the more you can customize them to work efficiently.