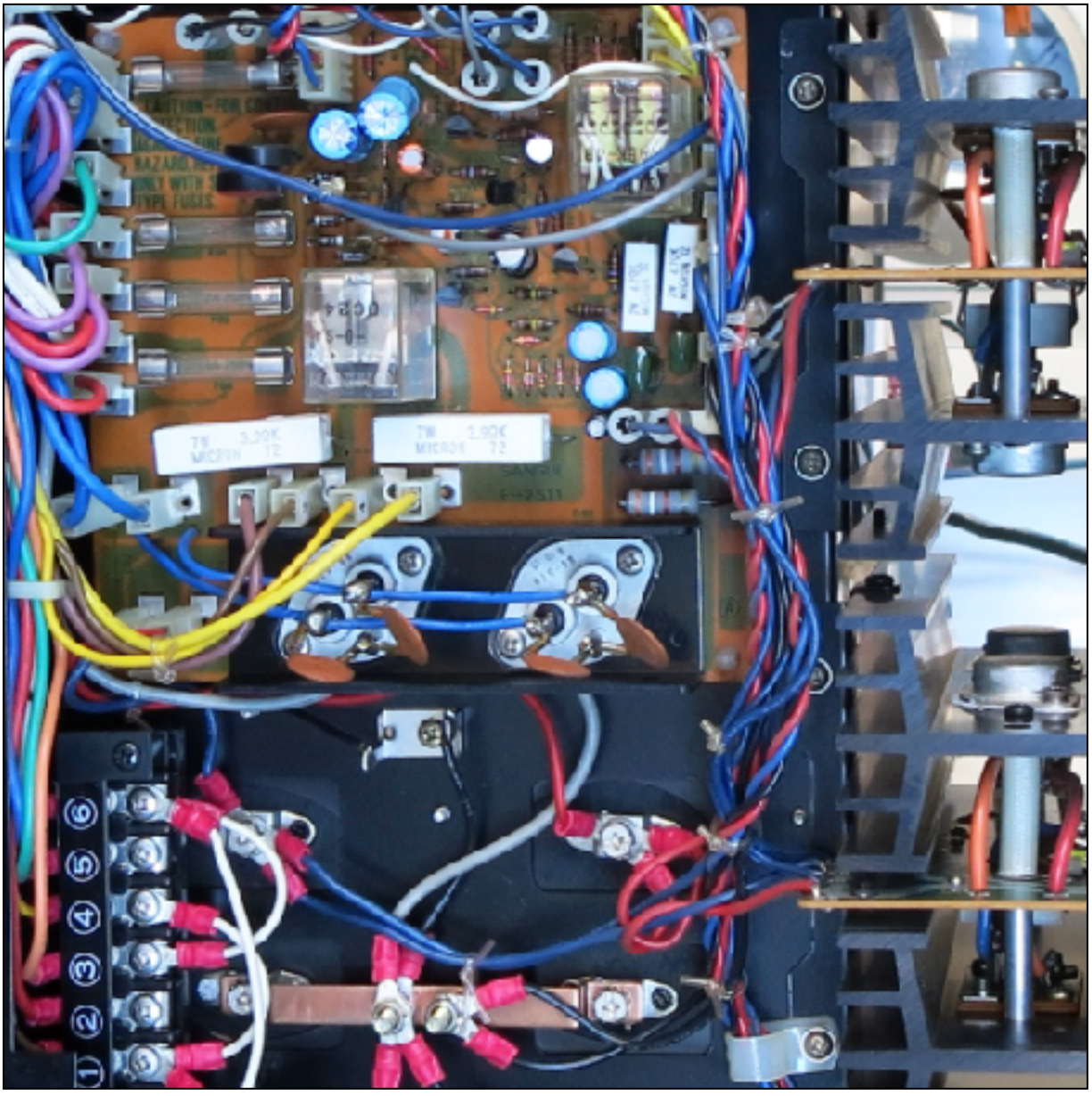
Core Concepts

Building Blocks

Techno-Plaza -Oct 1, 2023



Abstract

Most people are told "Computers are a box of switches" or "What's inside are just ones and zeros." This is deceptively over-simplified. Yes, the math approach used in computers is binary not tens-based. The math comes out the same but it takes a lot more digits to represent the same numbers. That's okay because computers are very fast. The gain is the binary system only has two states thus the ones and zero explanation. This matches up with the box of switches that also only have two states. Two states is easily handled by an electronic transistor switch, which can be made very small. The gap in the explanation is that most of these switches are used to maintain control of logic circuits, switch in and out whole bus systems, and performs math algorithms by controlling the state of the whole machine during each step. This binary math/control is ubiquitous because it simplifies the electronic circuits not because it is magic.

So to understand computing hardware we need to focus on building blocks not conspiracy theories. Logic gates, address and data bus access, and all the rest are controlled by the state of the whole machine meaning each step from pulling an instruction from memory, parsing it into opcodes and address schemes, pulling data from memory, acting on that data, storing the results, and moving on to the next instruction is done by setting control logic values to be either on or off in a register that can maintain the state of the machine during each step. Many steps later, something gets done. It's the many little pieces acting together during each instruction cycle that gets results. It's like a clock whose ticks are controlled by an escapement for each second (It takes a tick and a tock to get a second of time). Those seconds are cogged out into minutes and hours by gear reduction all synchronized literally within the machine. In a computer, adding two numbers requires a lot of little steps before the result ends up back in memory.

This paper looks at building blocks that are part of every computer and gadgets that perform various operations that could be controlled by a microcontroller or perform stand-alone functionality. The value in being aware of these circuits means you can build parts of a computer but also you can design unique functionality by using components from a computer. This is what makes hobby project so much fun to do.

In the Beginning

Within the guts of all computing devices that harbor a central processing unit, memory system, an I/O labyrinth of interconnectivity are numerous very simple constructs. Just saying this is much more complicated than the real silicon circuits themselves. What I'm talking about are logic gates that combine together to form multiplexors, encoders, priority encoders, decoders, adders, routing gateways, arithmetic calculations among the many specific purposed circuits. And I do mean specific. Computers are state machines. That means, each and every functional state has to be repeatable and distinct. It can be combinational logic or it can be timed or synchronous logic, but even timing is under distinct control to know the state of each operation. The fundamental chunks of a processor are input, output, memory, data path, and control all orchestrated with logic gates.

At the intersection of the real-world and the digital there are, of course, special electronic circuit designs that bridge the gap. Building a memory system, whether a single register or a bank of interlinked memory bits is a specialized circuit intersection with basic logic gates. Even these circuits are based on simple circuit elements. Once you go beyond the processor boundary past the I/O terminals, all bets are off. Electronic sensors and external controls are complex. Energy types like controlling RF and data delivery systems are extremely complex and the software to keep up with them equally complex but how a processor is implemented in circuitry is not. Let's look at the building blocks.

Logic Gates

Let me first say, there is an associated video that goes into detail as the companion to this paper called "Core Building Blocks" and can be found in the Tech Talks area of the Welcome to Design page of this website.

The basis of controlling digital functionality is accomplished by combining various "logic gates" to perform a function. This Combinational Logic, as it is known, consists of An AND, OR, XOR, and inverters. Their names describe their function and can be shown using a "truth table." AND-gate requires all the input signals (On or Off as a voltage and can be represented by a "1", "0", H, L, High, Low or true or false) to be On or present for the output to go High or On or at circuit design voltage level to be used by the next logic gate. The fact there are several terms for the same result creates confusion but remember, in the end, it's a voltage (5v for TTL) or the absence of that voltage that activates the logical outcome, which is also described likewise. The OR-gate is a nexus point that allows any High signal to produce an output High. OR-gates are connecting buffer nodes within a logical system. XOR-gate only allows an output when only one of the inputs at a time is present, in other words, one input is exclusive to all others. I have assumed up until now we are comparing one signal to one other but the functionality is same with multiple inputs to get one output thus a gating (filtering) effect. At first you might wonder where's the logic. Think of the AND-gate this way, "If this is true and that is true, then it follows that the outcome is true." If you're Greek, this all makes perfect sense. Same with OR and exclusive OR. The real question is how can these simple ideas amount to anything useful? Let's start with an additional circuit type called the D flip-flop. This small circuit has two states. The circuit has complementary outputs meaning it has two, one that tracks the input value and one that is always the opposite state. We place a High or Low at the D-input pin and when a clock signal is applied at another input, the output that tracks the D-input goes to that value called the Q-output. Therefore, the Q-bar-output will be the opposite state. The second output is just there as a convenience if an inverted value is used later in the circuit. The main point of the circuit is whatever value is clocked-in is stored in the circuit until another value is clocked-in at some other time. Now we have a way to create state in general for the whole system (up to this point).

So with gates to accomplish some logical decision making, we have a way to store the results in what's called a register. The number of these flip-flops needed for the register is matched to the number of bits we want to move around as data or addresses. The memory part of the processor is no more than a generalized set of registers whose circuit is optimized to scale up to vast proportions. Registers can be generalized outside of the main memory as buffers, places to hold data temporarily or collect data until needed. Registers can be used to set up a group of control signals where each of the Q outputs of the register go to other combinational logic for further define operations or just report current status of the processor. But notice, we have introduced a clock into the processor configuration. By controlling the clocking, we can control the state of the whole machine.

Design Factors

Besides logic gates and registers, we have mentioned memory and clocking. When designing a processing system or central processing unit (CPU), we have to settle on a word size in bits that each address in memory can hold. Part of the Operating System (OS) needs to know how big various data types are like integers, floating point numbers, strings and the like to determine how many words are needed to fit any particular data type into memory and a scheme to keep track of the address space if what's stored takes up more than one word size. Memory units are normally set up by 8-bits groups called bytes. If a design calls for 32 bit word size, that would mean there are four bytes stored at any one given address. This size chunk of a byte was chosen as the base because each ASCII defined letter, number, or punctuation can be held in one byte. Also, be aware that we have three items to keep track of to do anything useful with the memory - the address, the data, and the data type. The address bus can be any size depending on the addressing scheme to encompass the total volume of the memory space. That could be TeraBytes. The data size and its bus match the word size to efficiently stuff data into the memory at word boundaries.

Gathering all these design specifications together and turning it into clumps of combinational logic and specialize circuit bits is clearly going to become unwieldily. And at the core, is mainly a very organized bunch of switches stitched together into a set of states that do something useful like add, subtract, multiply, and divide not to mention branching, comparing, and reporting status and it is all done with a mass of little circuit components. The clocking is the way we get from on state to another no matter how small the steps are or how many steps it takes to accomplish any one given task.

Instruction Set Architecture(ISA)

Where all this circuit design to give us a functional processor, memory, and the software to organize it, is the ISA. We have to store our instructions in the memory so we need to consider word size. A Reduced Instruction Set Computer (RISC) matches each instruction length to the word size of memory. At this point, we are talking about a Central Processing Unit (CPU) and leave behind the simple logic designs. So let's say we have 32bit words. A typical design would have six of the most-significant-bits (MSB) devoted to instruction codes or opcodes. This is enough bits to cover all the types of instructions the CPU will need. The next three sets of five bits give us a register number to store two operands and a place to store the results. We have 10 bits left to subdivide functions or typically take advantage of bit-shifting to double or divide by a power of two or bitwise manipulation. Other instructions will enable branching to other memory locations to facilitate decision making as in an "if statement". All of this starts to beg a way to write code that turn bits (machine code) that can be understood by humans to load into memory as instructions to get the ball rolling. Each one of these bits are used to control the state of the machine. The data is store in another area of memory to be acting upon. Instructions are pulled from another place in memory and interpreted from the opcodes.

Each instruction bit pattern sets up a bunch of logic gates to pull data, place it into registers, move data operands into a circuit called the Arithmetic Logic Unit (ALU) that is yet more logic gates that can add, subtract, compare as AND or OR logic, increment or decrement to adjust memory locations the main one being moving to the next instruction in memory after the previous instruction completes. Be advised that each instruction takes several micro-steps to complete so the clocking controls are paramount. The point is, that up to this point we have been slamming together little electronic circuits that flip on and off and not one mention of binary counting or math operations.

The math part is really straight forward because binary representation of numbers is the same as the decimal system. It just takes a lot more digits to represent the same value. The gain in using binary numbers is they can be represented with only two states. As long as you have a one to count, you can always start from there and add more counts. The beauty of binary is we can place numerical data in memory as a series of “1s” & “0s." It looks just like logic gate manipulation information, which emphasizes the need to clearly separate control signals from numbers. Everything else is a matter of defining what a series of bits mean. For example the bits 01001000 or one byte, if known by programmers as string data interpreted as a capital H. Here again, complexity can be represented in various ways. The 01001000 bits can be split in the middle where 0100 = 4 and 1000 = 8 so 48 in hexadecimal or just read the whole binary value is 72 in decimal. The point being that any set of bytes can be pulled out in various representations but in the end a H is printed to a screen thus bits to intelligent information has taken place - yeah!

Since all these clusters of “1s” & “0s” only make sense if we as humans decide on a convention to establish what they mean. The standard for lettering was first implemented with the American Standard Code for Information(ASCII) which gives us one byte per symbol. How to get that representation onto a screen requires a bunch more digital logic gates!

Software Abstraction

I have to assume you now have a glimpse of how logic gates end up with “1s” & “0s” in memory of a CPU design. The gorilla in the room is going from machine code to something a Web Dev programmer writes to use this machine. It starts with the processor manufacturers settling on an ISA and then writing out how to go from what the logic does and code to represent it. Once that bridge is crossed, we have abstracted hardware into software. The manufacturer tells the world in a data spec for the device what the "assembly code" is to reach the instruction set operations. Then software developers pull together a "higher order language" that is easy to write and read that relates back to the assembly codes via a compiler or a software interpreter that converts your source code into assembly code, which is linked down to machine code. There is now a direct relationship between this higher order language, whatever it might be, down to the machine code that makes the computer sing.

Summary

Of course, this is a 70,000 foot look at what has become the most complex thing mankind has ever put together especially when you bring in the Internet and all the data centers in the world, but realize this - it's the meager little logic gates that are the bricks in the wall. It's software that the architect that makes it all beautiful. Check out the associated video (Core Building Blocks) for more details.