Sierra College **CIE-01**
Jim Weir **530.272.2203**
jweir43@gmail.com
www.rstengineering.com/sierra

Lesson 11

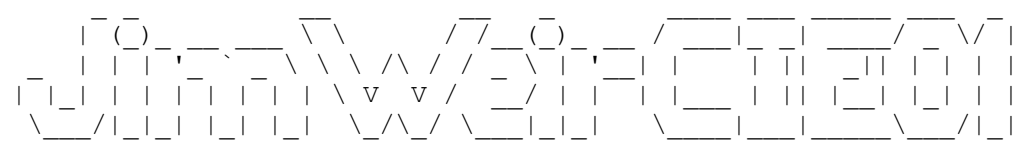
Elementary Digital Integrated Circuits
Number Systems

Digital Fundamentals: Unlike their analog colleagues that can have an infinite variety of input and output levels, digital circuits have only two states. We normally call these states "high" and "low", but we could just as easily call them black and white, up and down, north and south, giddyup and whoa, or any other diametrically opposed pair of words.

Digital circuits are therefore BINARY devices. In the binary world, there is no such thing as 2, or 147, or $3x10^8$. There is only 0 and 1 (or, as we said above, low and high, north and south …). How boring, eh? The old joke is that some programmers are so old, they can remember the old days when we only had 0, not even a 1.

The nice thing about only having two states is that you can have a data storage system that is blindingly fast. For example, the lower case letter "a" is kept in your computer as an 8 character digital "word" 01100001. Each of these 0 and 1 are called "bits" and the whole digital "word" is called a "byte". Thus, each letter and character in this document is kept in computer memory in 8 special locations. When we have a million of these locations, we are said to have a 1 Megabyte (one million bytes) document. A thousand 1 Meg documents are stored in a 1 Gig (one billion bytes) storage space, and so on. As a matter of fact, to get to this point (.) requires a storage space of 1405 bytes, or about a kilobyte and a half. By the time I get to the end of this lesson, I will probably have a 20 kb document, which is nothing when it comes to a hard drive that can store 200 Gb.

Of course, the bit twiddler weenies had to take it a step further and produce something called ASCII art:



And you can download the software to produce your own art at www.figlet.org .

Blindingly fast? With this 3 GHz. processor, it takes a little less than 3 nanoseconds to process an 8 bit byte, or about the amount of time it takes light to go 3 feet. Data processing has reached the point where the interconnect between the components has become the limiting factor in how fast we can compute.

Let's examine that 8 bit "a" word above. 01100001. What exactly does that mean. There is a convention called "ASCII" (**American Standard Code for Information Interchange)** that assigns a value from 0 to 255 to the keyboard characters plus some special characters like (00000001) and ÿ (11111111). How do we get to this magic number 255? The same way we get to the numbers in the decimal system. Let's looksee …

Number Systems: We are quite conversant with the decimal number system, whereby each time we move one space to the left, we've gone up by a factor of ten. Thus, the decimal number 247 really means 7 * 1 plus 4 * 10 plus 2 * 100. Two hundred and forty seven. Another way of explaining this same number (remembering that $x^0 = 1$ for any x) is to say that 247 is $(7 * 10^0) + (4 * 10^1) + (2 * 10^2)$. As a matter of fact, for any decimal number we can make a table (which, if not large enough, can be expanded to the left as as far as you need) that tells us what value each position in the number has:

10^7	10^6	10^5	10^4	10^3	10^2	10^1	10^0
10,000,000	1,000,000	100,000	10,000	1000	100	10	1
2	4	7	1	3	2	7	6

This number, obviously, is twenty four million, seven hundred and thirteen thousand, two hundred and seventy six.

Well, if this system works for ten (a "base ten" number system), we ought to be able to use it for ANY base number system.

Including a base 2, a binary system. Let's see what all that looks like:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
0	1	1	0	0	0	0	1

Let's see now, we have a 64 + 32 + 1 = 97 (more formally 97_{10} or 97 base 10). But ASCII decimal 97 (according to the chart at http://www.csgnetwork.com/asciiset.html) is simply the lower case character "a", which is where we started this discussion a few paragraphs ago. Note also that binary 11111111 is 128 + 64 … +1, or 255, which is where we run out of 8 bit numbers.

But look at the chart at csgnetwork again. It gives values for "Oct" and "Hex" in addition to "Dec(imal)". What are Oct and Hex? First guess, since Hallowe'en comes at the end of Oct(ober) is that somebody is going to cast a spell, or Hex you. Sorry, not so. More number systems.

Oct is simply the abbreviation for the OCTAL system of numbering, which is base 8. Thus, the octal values go:

8^7	8^6	8^5	8^4	8^3	8^2	8^1	8^0
2,057,192	262,144	32,768	4096	512	64	8	1

The reason that the octal system was invented was to make calculations on the first series of microprocessors easy. The first microprocessors were 8 bit machines and using base 8 made the calculations fairly simple. Somebody please explain the joke as to why programmers always confuse Hallowe'en with Christmas.

But now that we have 16 (and 32 and the current state of the microcomputer art 64 bit) bit machines, we needed a slightly larger numbering system which we got with the HEXADECIMAL (base 16) number system. However, now we have a problem. We had arabic numbers for the base 10 system (0-9) and arabic numbers for the base 2 system (0-1), but now we've run out of numbers. Instead we use Capital Letters for the numbers between decimal 10 and decimal 15. A (10), B (11), C (12), D (13), E (14), and F (15).

16^7	16^6	16^5	16^4	16^3	16^2	16^1	16^0
268,435,460	16,777,216	1,048,576	65,536	4096	256	16	1

Gets real big real quick, doesn't it. Wow, now that we've got a numbering system that will take us out past 268 million in 8 bits, what's next? Easy, the DUOTRIGESIMAL (base 32) and QUADROSEXIGESIMAL (base 64) numbering systems. You think the hex system gets big quick? 64^7 is 4.398… ***TRILLION.*** To paraphrase Senator Everett Dirkson, a trillion her, a trillion there, pretty soon you are talking about real money.

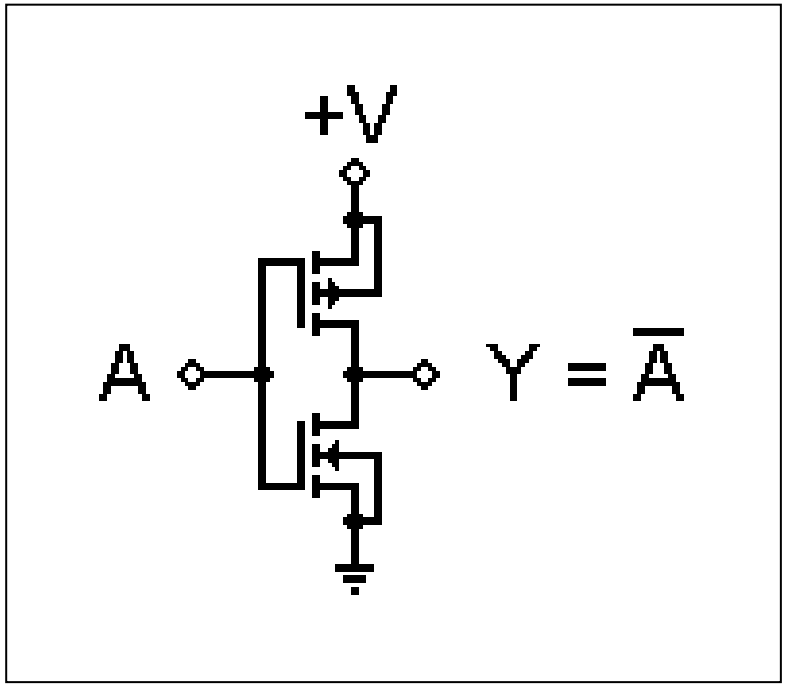
Elementary Digital Devices:

Right now, we don't worry about what is inside of the package or how the transistors inside are connected; we will concern ourselves strictly with what happens at the output for various inputs. And, while there are half a dozen common digital families, let's concentrate on what is called "CMOS".

CMOS -- Complementary Metal Oxide Semiconductor. Or, in other terms, P and N type MOSfets in various configurations. Not the fastest family, nor the widest number of different configurations, but by far the most power saving and least expensive. Here we see a rather simple device with both P (upper) and N (lower) channel FETs in a logic arrangement.

Let's get some nomenclature for digital devices. "A" is the input. What we want to know is what happens at the output for the only two possibilities of A -- high or low.

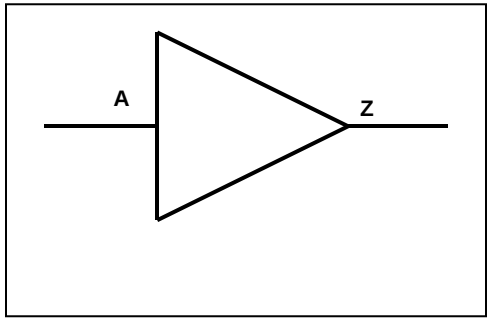
When the input is high (i.e. very near +V volts), the upper P channel device is off and the lower N channel device is on. With the output thus cut off from the +V supply by the off P channel and directly grounded by the on N channel, the output is ground, or zero volts, or low. Conversely, when the input is low, or near ground, the lower N channel device is off and the upper P channel device is on (conducting) and the output is high, or near +V volts.



We generally have what is called a "truth table" in the data sheet for the device in question. The truth table tells us what will happen at the output for any combination of inputs. The truth table for this simple device looks something like this:

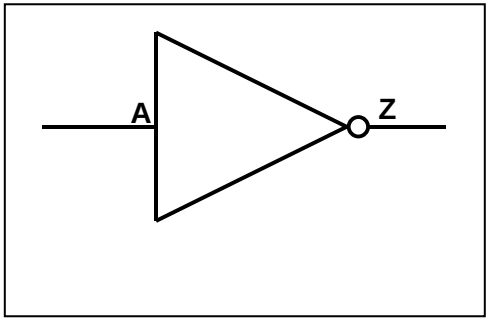
A	Y
0	1
1	0

The simplest device that we have in the digital world is a BUFFER and it's companion the INVERTING BUFFER. While it may seem kind of silly to have a device that does nothing more than repeat or invert what is at the input, there are a lot of reasons for using this buffering device -- mostly at the input of a circuit to isolate the circuit from the outside world.

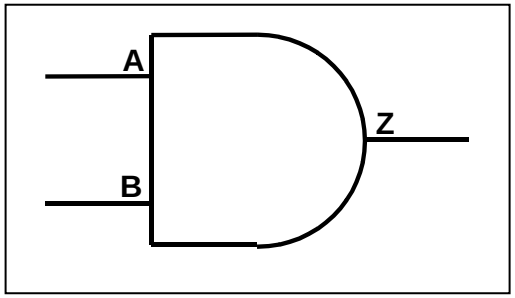


Rather than go into detail on where you might use a buffer, let's look at these two buffers. The one on the top is the plain buffer … whatever logic level is present at A will be repeated at Z. The one on the bottom is the inverting buffer. Whatever logic level is present at A is inverted at Z … if A is high, Z is low; if A is low, Z is high. Note that the circuit at the top of this page is an inverting buffer.

The little circle or dot at the output on the schematic symbol tells us that this is an inverting circuit. In general, all simple "gate" type logic have both inverting and noninverting versions.

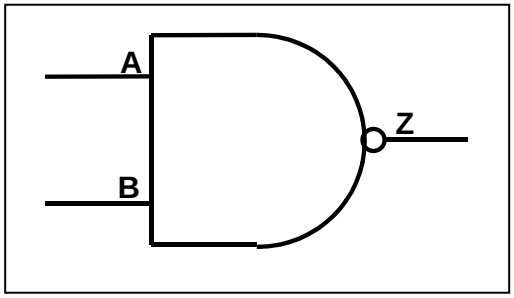


The basic gate, the gate from which all other gates may be made, is the AND/NAND couple. This gate simply says that if both inputs are high, then the output is activated. If either or both inputs are low, the output is deactivated.



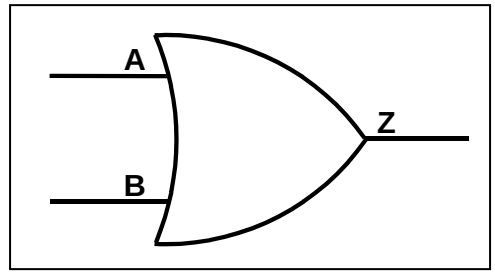
What does "activate" mean? Let's give an example using the AND gate at the top of this page. If both A *and* B are logic high, then Z is also logic high. If either or both A or B are low, then Z is also logic low.

The NAND gate (shown to the right with the "bubble" on the output) is the exact inverse of the AND gate. If both A *and* B are logic high, then Z is logic low. If either or both A or B are low, then Z is logic high.



Where might we find one of these devices? How about a washing machine? If the "A" door is shut AND the "B" timer is on, then turn on the "Z" motor. If either door or timer are "low", then no motor.

Another typical gate is the OR/NOR couple. Somewhat similar to the AND gate, the OR gate simply says that if either of the two inputs is high, then the output is activated.

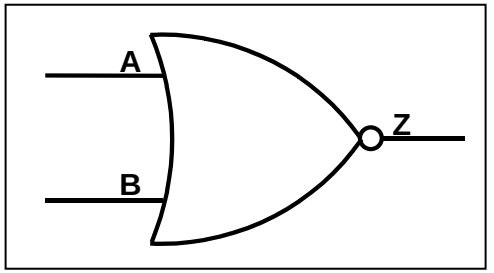


Thus, if either A *or* B is high in the OR gate, then the output is high.

Conversely, if either A *or* B is high in the NOR gate, then the output is low.

Uses? How about a burglar alarm? If either the A front door has been opened OR the B window has been opened, then sound the Z alarm.

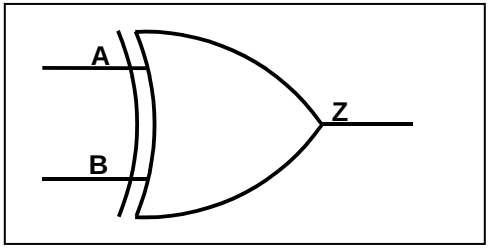
Take it a step further. Run the Z of the burglar alarm OR gate into the A input of an AND gate and the "arm" switch voltage into the B input of the AND gate. Then we get the alarm if either the front door has been opened or a window has been opened AND the system is armed.

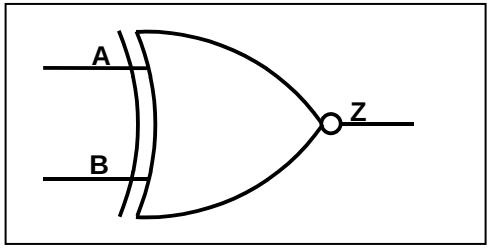


One problem with an OR gate is that we will get an output if A or B is high, but we will also get an output if both A and B are high. We get around that with what is called an exclusive or (XOR) gate.

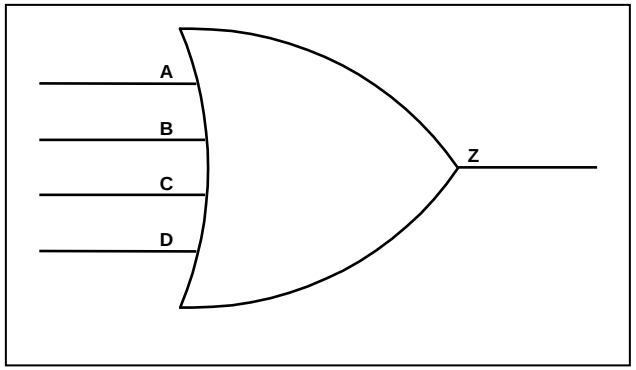
For the XOR, if either A or B is high, then Z is high. If both A and B are high or both A and B are low, then the Z output is low.

And, just like with all the other gates, the NXOR gate simply inverts the state of the Z output.



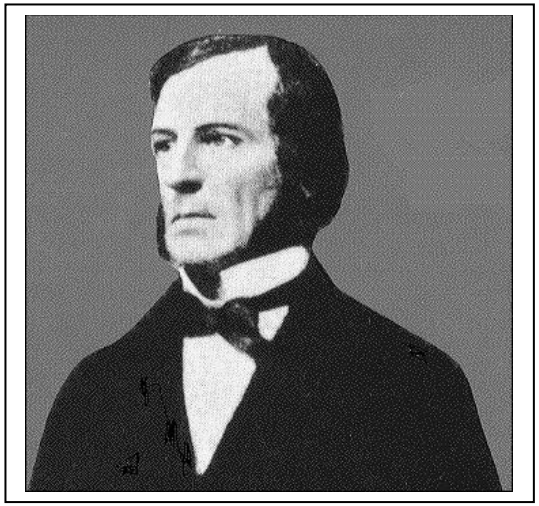


This is not to say that we can't have multiple inputs to our gate. Here is a perfectly legitimate 4-input OR gate. Where might we use such a device.



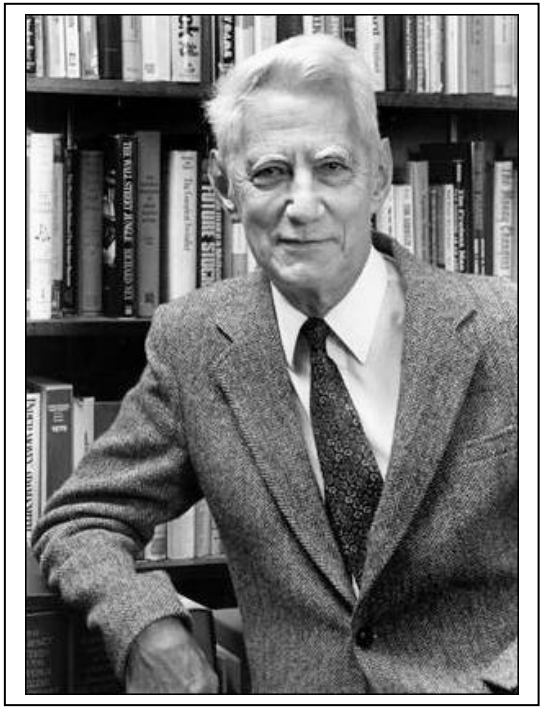
Back to the washing machine. If the water level is too high OR the door is open OR the water is too hot OR if the timer isn't on, then turn the Z water off.

Who thought up all this foolishness about logic and such? A fellow by the name of George Boole, who invented Boolean algebra in 1847. A brilliant man, he taught himself Latin as a child and was translating Latin poetry before his teenage years. By the time he was 13 he was fluent in Latin, English, German, French, and Italian. At 16 he became a high school teacher, and at 20 he opened his own school.



Unfortunately, he died of pneumonia at 49, having walked to his school in a downpouring rain and taught all day long in wet clothing -- thus showing once again that genius and common sense do not always go hand in hand.

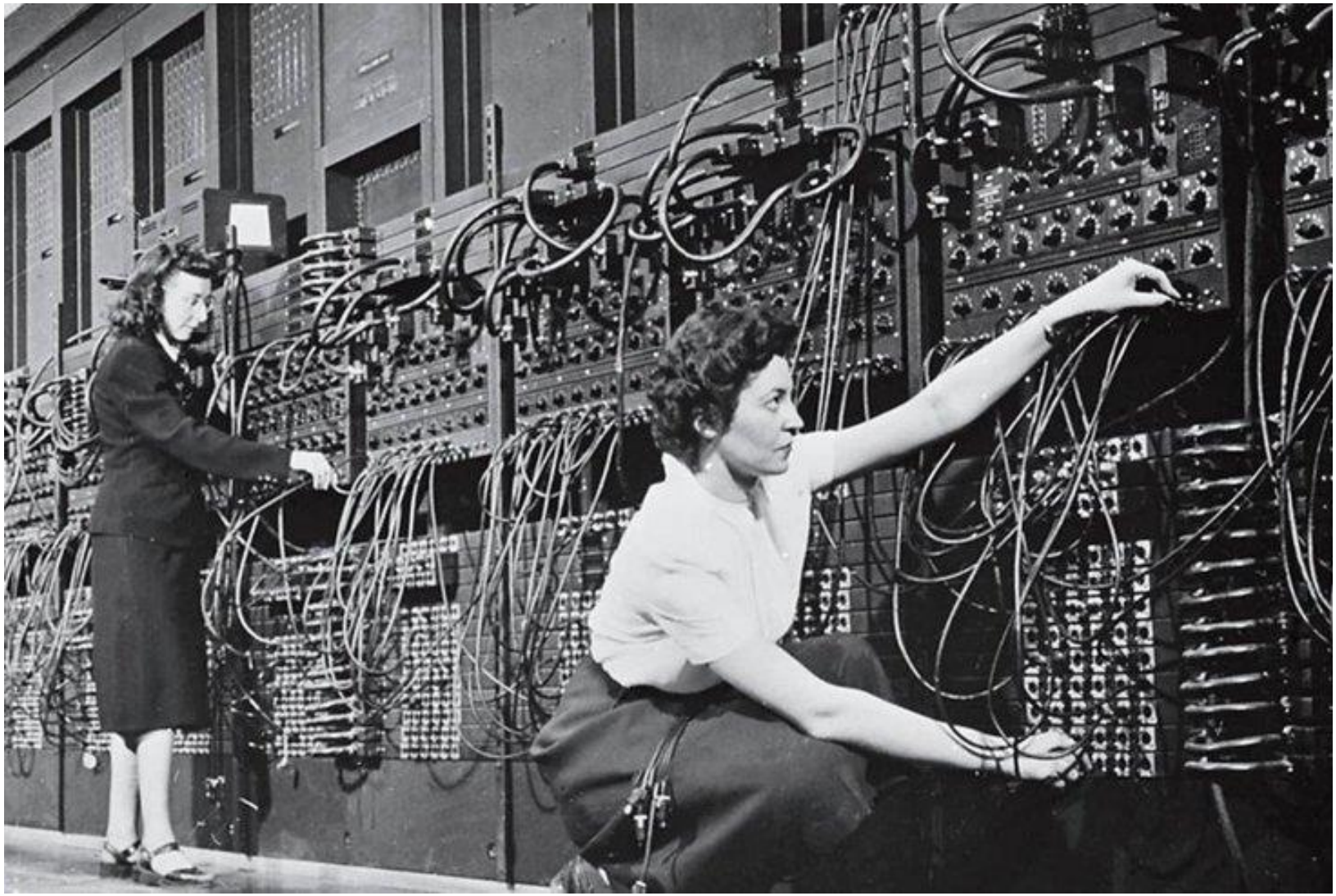
Fortunately, Boole had one student, Claude Shannon, that took his theoretical algebra and converted it to electrical machinery, most notably the computer. You can get an insight into Shannon's mind with his most famous quote:



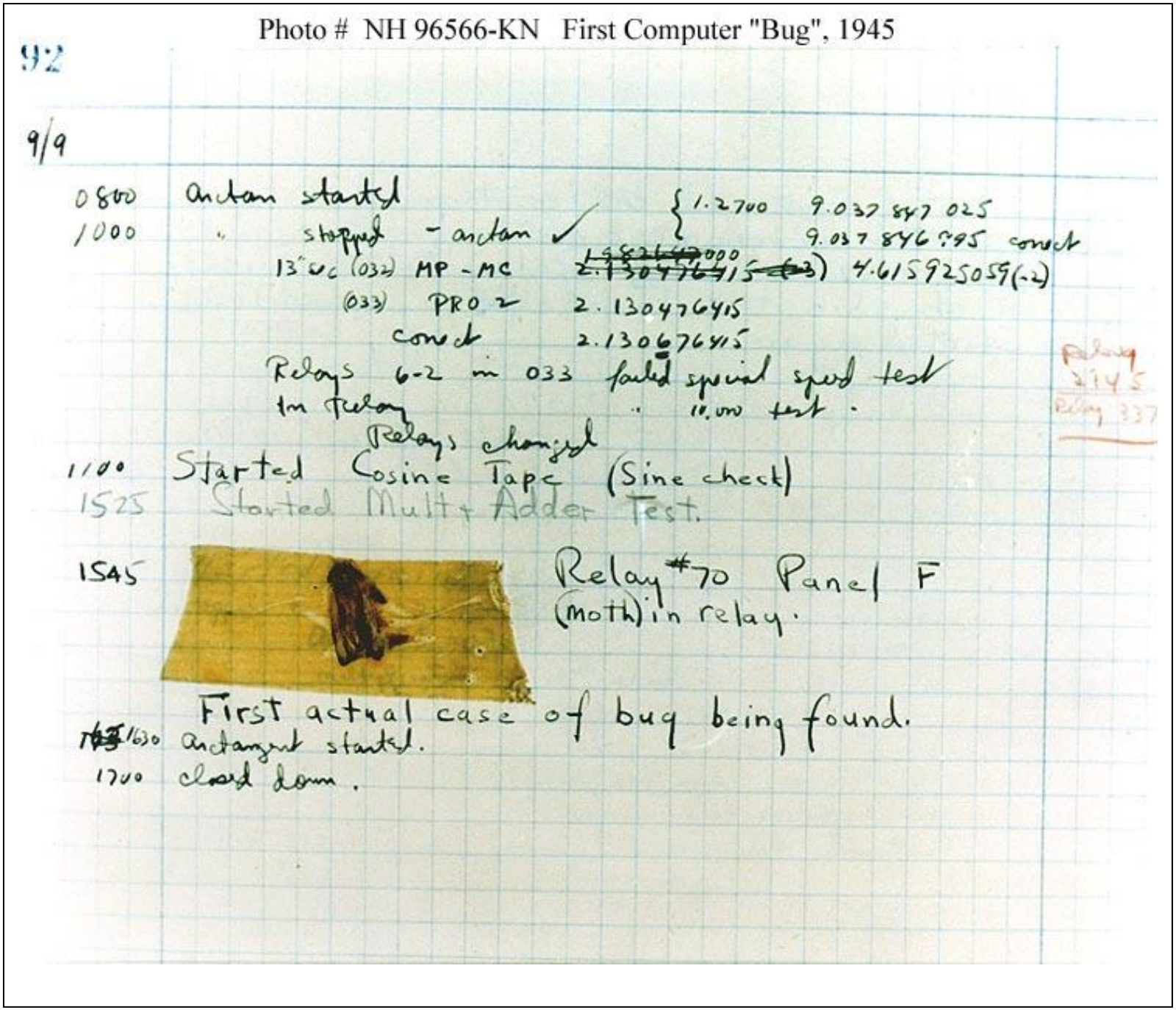
"I visualize a time when we will be to robots what dogs are to humans. And I'm rooting for the machines."

And what came of all this? ENIAC (Electronic Numerical Integrator And Computer) is widely believed to be the first electronic computer. Built for the Army in 1943-1946 and originally intended to produce artillery calculations, it was actually first used to calculate the equations for the first hydrogen bomb.

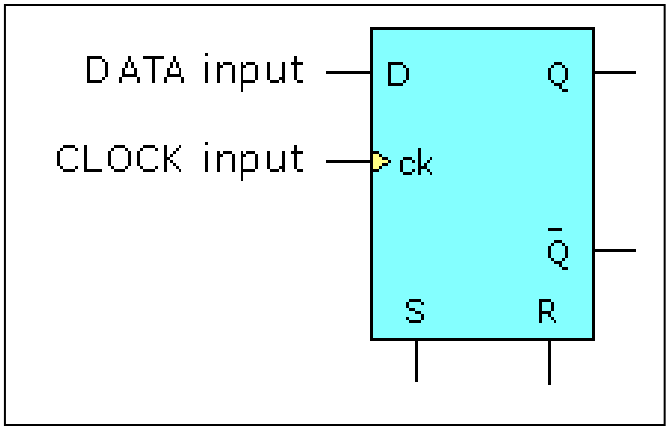
18,000 tubes, 5 million hand-soldered joints, ten miles of wire, it weighed 30 tons and required 150 kilowatts of power.



And what could bring this machine to it's knees? In 1945, a poor moth got stuck between two contacts of a relay and became the first computer "bug". The technicians said that they had "debugged" the computer.



Finally, after all this horsing around with gates, we come to a circuit made up of a bunch of gates. This is called a "bistable multivibrator", or more colloquially, a "flip-flop".



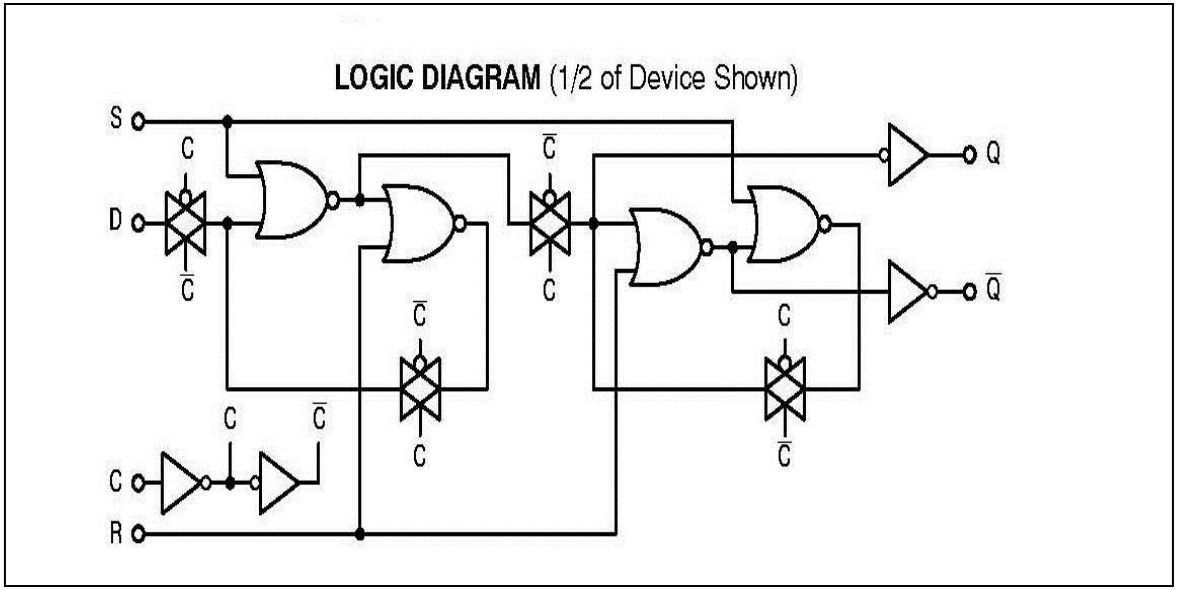
The flip-flop is interesting in that it FORMS THE BASIS FOR ALL MEMORY.

Here is how the flip-flop works:

a. Presume for the moment that both R and S are at logic low (0 volts, or ground). Presume that Q is high and /Q (not-Q) is low.

b. Presume D is low. With a switch, put a high on the Ck (Clock) input then immediately return the clock to low.

c. What happened is that immediately when the clock went high, it transferred whatever level was at D directly to Q. In this case, it put a low on Q and by definition of not-Q (/Q) put a high on /Q.



d. Now no matter what happens to D (it can go from high to low and back again a bazillion times) Q and /Q do not change. It is only when Ck goes from low to high that the Q and /Q can change. This "memorizes" whatever D was the last time Ck "transitioned" from low to high and it matters not from that point on whether Ck goes back low, whether the data changes on D, Q and /Q remain where they were during that low to high transition of Ck. It "memorized" the state of D during the transition.

e. S(et) and R(eset) are over-rides to the Data and the Clock. When S is high, no matter what D and Ck are doing, Q goes high and /Q goes low. When R is high, no matter what D and Ck are doing, Q goes low and /Q goes high.

f. In an apparent lapse of "logic" we have an oddball case when both R and S are high simultaneously. Both Q and /Q **BOTH** go high. By the strict rules of logic, /Q is always the INVERSE of Q and the designers of this part took a bit of liberty with the rules.

Each flip-flop is one "bit" of memory, and 8 f-fs make one 8-bit byte of memory. 8 billion flip-flops make up one GIGAbyte ("gig") of memory, so your 2 gig computer memory has 16 billion of these little rascals flipping and flopping all day long.