Digital Systems

Introduction

Mankind has gone through many "ages"—periods of time when civilization has progressed through a particular new idea or technology. The bronze age, the iron age, the industrial age, the age of discovery are examples. Today we're in a different age. Some would say the computer age, some would say the information age. I prefer the latter, because information is the driving force and includes computers and digital technology.

What is information? It's knowledge or ideas or data that can be communicated. But, we've done that for 5000 years, so what makes now so special? The answer: we now have computers and technology to represent, organize, and manipulate this information with unimaginable efficiency. Ten years ago, an airmail letter sent from New York to Paris might take a week to arrive. We thought that was pretty good. Today we call that "snail mail". Today we expect instant delivery of almost everything. Email, on-line shopping, real-time weather satellite photos, problem solving using a thousand computers world-wide: this all exists today.

This all exists today because information representation, signal processing, digital logic, communication protocol, computer design, storage and retrieval systems have been brought together through a single technology: digital computers. And this technology has been brought to us through engineering—some electrical engineering, some computer science. The fields are now so intertwined that academic departments often can't decide what to call themselves. "Computer engineering" seems to be a current favorite. In this section we'll look at some of the more fundamental ideas of this digital revolution.

Representation of information:

Roughly, information is represented in one of two ways: **analog** or **discrete**. And, which of these ways is used depends on the nature of the information. Before we define analog or discrete, let's begin with a problem. Suppose we want to measure the voltage across two terminals. What sort of instrument might we create to do that? Fortunately we remember that Hans Oersted in 1819 discovered that a magnetic needle changes orientation in the presence of a coil having an electric current. And its change of orientation is proportional to the strength of the current. So, if we know the resistance R of the coil, we can relate the voltage E to the current I using Ohms Law E = IR.
So we build such a gadget, calibrate it with a known voltage, and presto we have a voltmeter. (Actual voltmeters are built slightly differently. The magnet is fixed, and the coil is attached to a movable, non-magnetic needle. But the principle is the same.)

What we've done is we've devised an instrument whose needle position is "analogous" to voltage. That is, in this instrument, voltage is represented by angular position. So, because of the physics and materials at hand, it was natural to create an analog representation of voltage.

Let's consider another problem—suppose we want to design a device that will tell us whether or not we've left a garage door open. All we want to know is open or closed? Most likely we would attach a mechanical switch to the garage door that would be "open" when the door is closed and "closed" when the door was open. We could then use that switch to control a light which is placed in a convenient location. Here our instrument is "discrete". It's output is one of two states; the light is on or off. A partially closed garage door would be considered open. So we've designed an information system that encodes just some features of the garage door. The system does not provide an output which is analogous to the position of the door. Only if it's all the way closed or not. So this is fundamentally different from an analog system.

Position of a needle on a meter, level of water in a rain gauge, amount of electric charge in a capacitor, orientation of a gear: these are all analog information devices. Each has a continuum of values with which to represent or store information. So is a photograph—with a continuum of tones and exceedingly fine spatial resolution.

Contrast that with an on/off switch, or a digital clock, or the number 12345. Each of these has a discrete number of states or possible values.

Is one better than another? Maybe. Suppose it rains, and we want to know how much. Our rain gauge has that data in the analog form of the height of water We take a meter stick and measure this height. There are several problems. First, how accurately can we measure the height? To a centimeter? To a millimeter? Probably not much better than that. Further, because we would probably have to interpolate between meter stick markings, there would be some judgement required. So three people taking the same measurement might come up with three slightly different values. Second, how representative of the rainfall is the height of the water in the gauge? Could some of the water have evaporated between the time it stopped raining and the time the measurements were taken? So, even though the rain gauge might have done a very good job in representing rainfall, "instrument drift" and measuring complications reduces its effectiveness.

Measurement problems are a major concern in analog instruments or storage devices. On a good voltmeter, the scale behind the needle has a mirror. Why? Because the needle lies a few millimeters in front of this scale, and depending on the angle at which you try to take a reading, you can get different answers. Using the mirror you can align your eye so that the needle and its image are coincident. In this position, you know you are viewing the needle at right angles to the scale, so your reading will contain no **parallax** error.

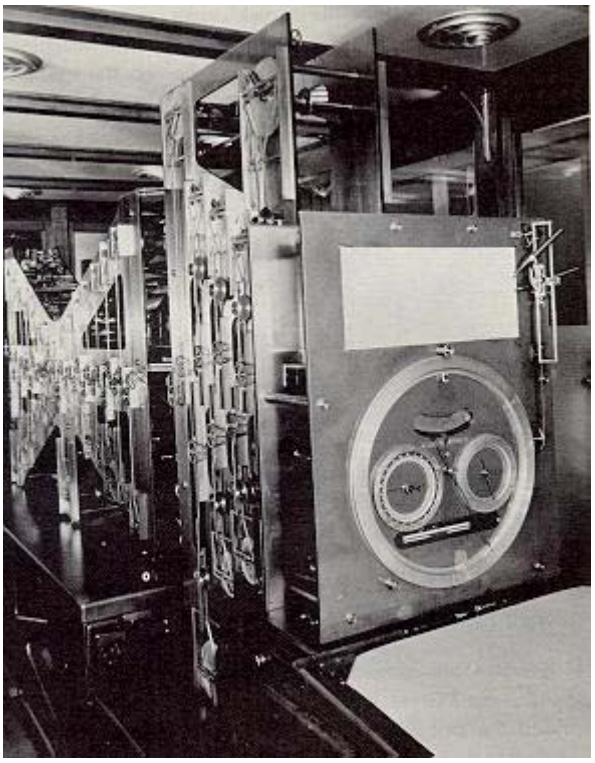
How good a measurement can we get of, say, a length which is analogous to the value of some parameter? Let's just say that spatially we can resolve 0.1 mm. Suppose we need to do better than that. What can we do? One solution is to "amplify" the distance. This can be done optically, as with a microscope, or mechanically with a micrometer. (There are more sophisticated ways as well.) A micrometer amplifies

distances using a cylinder with very fine threads. One complete turn of this cylinder might move it 0.1 mm. But, suppose the diameter of the cylinder is 10 mm, then the circumference is approximately 30 mm. If you can resolve 0.1mm rotation of this cylinder, you will have improved your measuring ability by 300.

And then there's the problem of data loss. The vinyl phonograph record provides an example of an analog information storage problem. Here, music is represented as a wiggly groove. A phonograph needle tracks the wiggles in the groove and produces an electrical signal which can be converted to sound. After some time, however, the needle begins to wear the groove—especially the edges in regions of high oscillation. The result is irretrievable loss of information—in this case, loss of the higher acoustic frequencies.

These are some of the negative aspects of analog systems: instrument drift, measurement, problems, irretrievable loss of information. But there are some positive sides as well. When one wants to simulate some process, say the behavior of a differential equation, it is very natural to express that equation using an analogous model. As a crude example suppose we want to deduce the distance a car travels in time T, given that it's speed constantly changes. In mathematical terms, you want $D = \int_0^T v(t)dt$ An analog model to this problem might be for us to pour water into a container at a rate which is representative of the car's speed v(t). At our final time T, we simply weigh the container to deduce D. The rate of our pouring water is an analog to the car's speed. The amount of water is an analog to the car's speed integrated over time.

A more important example of a process simulator is the Great Brass Brain built in 1910 and used until 1966--a 2500 pound machine to determine tides. After setting 37 dials, one could use this assemblage of cams, gears and shafts to produce tidal tables by turning a crank—forward to predict forward in time, backward to predict backward in time. The gears represented the oscillatory effects of the sun, moon, and planets on the tide. No power failures. No disk crashes. And a lot better than the alternative—adding together hundreds of sines and cosines by hand.



Another analog example is Vannevar Bush's "differential analyzer"—a machine of gears, cones, disks, and shafts which analogously represented the mathematics of the trajectories of

projectiles. During WWII, this machine churned out artillery firing tables—an important contribution to the war effort.

Finally, there is that most-pervasive-of-all analog computers—the slide rule. For those of you who are too young to know, a slide rule is a pair of sticks—one sliding against the other--that gives you the ability to carry out multiply/divide calculations. Before the advent of digital computers in the late 1950's, the slide rule was the computing instrument of choice for all engineers and scientists. How did it work? The principle was simple: numerical value can be analogously represented by distance.

To help clarify the idea, suppose we wanted to develop a slide rule to add and subtract. Then, all we would need would be two meter sticks. To add two numbers, say 2 +3, the distance corresponding to 2cm on one stick would be added to the distance corresponding to 3cm on the other stick. This would yield a total distance of 5cm on the first stick, i.e., 2+3=5. That's how the slide rule works, except that it allows one to multiply and divide. How? Recall that multiplication can be effected by adding logarithms. So, if our meter sticks were marked with numbers whose spacing represented the logarithms of the numbers rather than the numbers themselves, then adding lengths would correspond to adding logarithms, i.e. multiplying. For dividing, we would subtract the distance of one logarithm from another. Pretty slick! The inset illustrates the calculation of 2*3 = 6, i.e., log(2) + log(3) = log(6). Note the logarithmic spacing of the numbers.

These devices all represent information as values of continuous parameters: needle deflection, amount of water, rotational position of a gear, distance on a slide rule. And these parameters represent the analogs of mathematical or physical processes. But we are now discovering that it is cheaper, more accurate, and more efficient to represent processes and information discretely or digitally—even though, in the real world, most processes are continuous. The same is true for the mathematics of continuous processes. We now find it preferable to develop discrete algorithms to approximate calculus, probability, and differential equations.

However, there is a branch of mathematics that deals with discrete variables: the mathematics of logic. Here, variables can take on one of only two values: true or false, 1 or 0. As we will see, logic will play an incredibly important role in discrete information systems.

We've already talked about devices to hold or represent analog information--shaft rotation, amount of water in a container, electrical charge on a capacitor. What about devices to hold discrete information? That is, what can we use to unambiguously represent one of a finite number of states? What devices can you think of? A relay or switch. Two states. It's either open or closed. A pit or dent (as on a CD). Two states. It's either there or it isn't. Magnetic polarity. Two states. N or S. Current in a transistor. Two states. Conducting or not-conducting. A light. Two states. On or off. Is there anything odd about this collection of devices for representing discrete information. Yes. They all have only two states. In fact, one is very hard pressed to

think of a device that can manage anything other than two discrete states. There are a few "tri-state" electronic elements, but not much else. Can you think of any?

The bottom line is that, if we're going to represent information and processes discretely or digitally, it appears that we'll have to do it with two-state devices. So information stored will be stored as **binary**, i.e., as one of two states. The good news is that binary is easy to detect. A light is either on or off. There is no such thing as a partially-on light. The bad news is that, since there are only two states per device, it will take many such devices together to represent the values of most pieces of information. A string of 10 devices will have $2^{10} = 1024$ possible states As an example, to represent time to the nearest second we would require 16 devices—4 devices to represent each of the 12 hours, 6 to represent each of the 60 minutes, and 6 to represent each of the 60 seconds. All this to replace an hour hand on an analog clock.

Actually, whenever we state or write down a value of a parameter, continuous or otherwise, we do so using a discrete representation. We write it using our decimal numbering system. Pi is 3.14159; the distance between Chicago and Baltimore is 749 miles; it rained 1.02 inches yesterday. It's a convenient way to express a value to some level of accuracy. Without thinking, we automatically interpret 749 as $7*10^2 + 4*10^1 + 9*10^0$. That is, each of the digits represents a power of ten. And within each digit are 10 possible states: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. That's how we arrive at 749. And from now on, we should really write 749_{10} to indicate that the number is expressed in decimal or base 10.

In a binary system, the representation is similar, except now we have only the states 0 and 1 to work with. That's all our devices can accommodate. So, with a string of 10 binary devices to represent the number of miles between Chicago and Baltimore, we would have 1011101101_2. Again, we have to assign the subscript 2 to indicate that the expression is in base 2. After all, 1011101101 is a perfectly good decimal number. So, again we clarify it with the subscript. This should be interpreted as $1*2^9 + 0*2^8 + 1*2^7 + 1*2^6 + 1*2^5 + 0*2^4 + 1*2^3 + 1*2^2 + 0*2^1 + 1*2^0 = 749_{10}$. Well, expressing 749 is a lot easier than expressing 1011101101. that's because there are far fewer digits in decimal representation than there are in binary representation.

Further, if I asked you to store the number 749_{10} as a collection of 10 binary digits, it would take you a long time to figure out which binary digits would be 1's and which would be 0's. That's because there's no convenient way to convert decimal digits to binary digits. To do that, we need another numbering system—one that has about the same information content as a decimal digit so it will be easier to write down or remember, and one that will easily convert into binary digits so we can correctly store it in binary storage devices.

Actually there are two such numbering systems: octal and hexadecimal. In octal there are eight states per digit; in hexadecimal there are sixteen states per digit. In "hex", the values are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. So, you can get weird looking numbers such as $2ED_{16}$. In fact, $2ED_{16} = 749_{10}$. So, why is octal and hexadecimal so

much better than decimal for expressing binary information? The answer is that both are number systems whose bases can be represented as 2^n. For octal, n = 3; for hexadecimal, n = 4. What this means is that a single octal digit can be expressed precisely as three binary digits. So, our binary representation of 749_{10} can be grouped for clarity as $1\_011\_101\_101_2$, and each triplet consists of exactly one octal digit—in this case 1355_8.

To help you see how these systems relate to one another, here's a table of the first 17 numbers in each of the bases.

decimal (10)	binary (2)	octal (8)	hexadecimal (16)
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10

Note that the right-most octal digit is always the same as the right-most three binary digits.

OK. So, now we have a system for representing information in binary. And, in principle, we can imagine actual devices to store that information. What about manipulating it, or performing operations on it?

Whereas binary representation is a nuisance in terms of communicating values, it becomes a blessing when it comes to performing calculations. Suppose we want to add two numbers together A + B one digit at a time. In binary, there are only four possible states for each digit and two possible states for the outcome. We can completely enumerate the operation of addition with an input/output table. For one digit, the sum of A + B = R(esult) plus a possible "C(arry)" to the next digit. Here's every possible combination of inputs and their outputs:

inputs		outputs	
A	B	R	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Let's pause on this binary addition problem for a minute, and return to a topic I alluded to a few paragraphs above: the mathematics of logic. Logic concerns statements or events which can be represented as combinations of discrete states: true or false, or 1 or 0. Here's an example situation. If Carlos and Mary are at a restaurant, they eat pizza if and only if Sam is not at the same restaurant. Another way of stating this is in terms of a logic table. We define two logical variables A and B as inputs, and one logical variable C as an output.

A = Carlos and Mary are at a restaurant—true or false?
B = Sam is at the same restaurant—true of false?
C = Carlos and Mary eat pizza—true or false?

inputs		output
A	B	C
0	0	0
0	1	0
1	0	1
1	1	0

Here, a "1" signifies the condition is true; a "0" signifies the condition is false. So the third entry states that Carlos and Mary are at a restaurant; Sam is not at the restaurant; the result is they eat pizza. In all the other situations, Carlos and Mary do not eat pizza. This is an example of mathematical logic. One can have lots of inputs and lots of outputs, but the idea is the same. For every combination of true-false inputs, there are specified true-false outputs. And all these combinations can be expressed in terms of a logic table. Notice we did the same thing for our binary addition table.

George Boole in 1854 linked arithmetic, logic, and binary number systems by showing how a binary system could be used to simplify complex logic problems. Boole recognized that any logic problem could be simulated using a combination of three elemental logic **connectives**: AND, OR, and NOT. The logical properties of these connectives are defined in terms of the logic tables below. We also include a symbol for

each connective, so we can produce diagrams of logical situations. Incidentally, in electrical engineering, these connectives are often called **gates**.

AND

inputs		output
A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

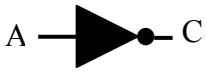
OR

inputs		output
A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

1

NOT

input	output
A	C
0	1
1	0



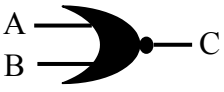
Other connectives can be created from these primitives. NAND (not AND) and NOR (not OR) are particularly useful.

NAND

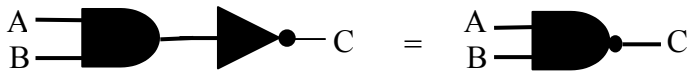
inputs		output
A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

NOR

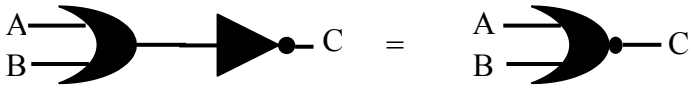
inputs		output
A	B	C
0	0	1
0	1	0
1	0	0
1	1	0



Notice that a NAND connective can be produced by a combination of an AND and a NOT, i.e.,



Similarly a NOR connective can be produced by a combination of an OR and a NOT:



Notice that a small black dot following a symbol signifies NOT. Look at the NOT, NAND, an NOR representations.

We can conjure up other connectives, such as XOR (exclusive OR) and XNOR (exclusive not OR) as well. What would their logic tables look like? Try creating such connectives using just AND, OR, and NOT elements.

Boole created an entire mathematics using these ideas of logic. Today we call it Boolean algebra. Using "*" to represent AND, "+" to represent OR, and "'" to represent NOT, the rules are as follows:

AND rules	OR rules
A*A =A	A+A = A
A*A'=0	A+A'= 1
0 *A =0	0 +A =A
1*A =A	1 +A = 1

Associative/commutative/distributive rules

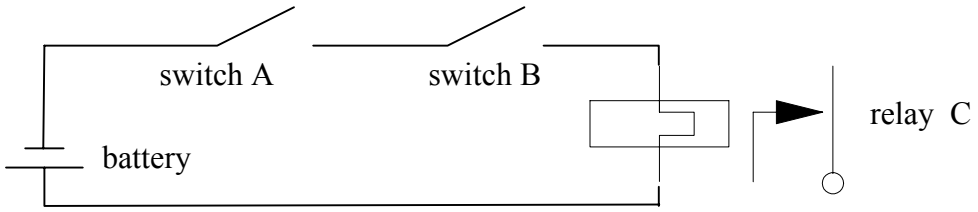
A*B = B*A
A+B = B+A
A*(B*C) = (A*B)*C
A+(B+C)= (A+B)+C
A*(B+C) = A*B+A*C
A+B*C = (A+B)*(A+C)

De Morgan's theorems

A'*B' = (A+B)'
A'+B' = (A*B)'

So, that's the mathematics of binary logic. Not so difficult.

The next step came in 1938 when Claude Shannon demonstrated that any logic problem could be represented by a system of series and parallel switches, and that binary addition could be done with electric switches. That is, AND, OR and NOT connectives could be implemented in hardware. Consider the following electrical circuit:

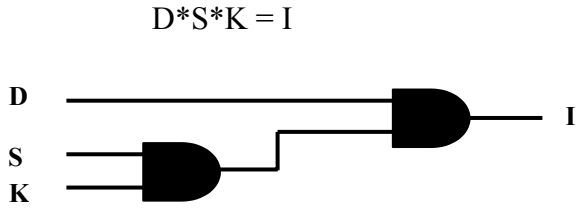


Here, relay C closes, if and only if both switches A and B are closed. If "closed" for the switches and the relay are interpreted as 1's (as opposed to 0's), then this electrical circuit represents an AND. What about an OR circuit? Study the logic table for an OR. Can you invent a circuit?

Now that we know the rules of Boolean algebra and we know that we can construct devices which emulate connectives, we should be able to piece together circuits that represent logical situations. Let's look at a few examples.

First, a control system. For example, a car will start (I = 1) if and only if the doors are locked (D = 1), the seat belts are fastened (S = 1), and the key is turned on (K = 1). Then the logic table and circuit would be constructed as

inputs			outputs
D	S	K	I
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1



How did we arrive at the expression D*S*K = I? We observed all the combinations of inputs that yielded I = 1. In this case there was only one. The result I = 1, then is equivalent to D = 1 and S = 1 and K = 1, or D*S*K = I. Note how the two AND gates are connected to form the equivalent of a three-input AND gate—which is really what D*S*K=I requires.

Next, a voting problem: Lois is elected chairman of a committee if and only if she gets a majority of the votes from the members A, B, and C. First let's write down the logic table corresponding to this situation. A "1" is a vote for Lois. Lois being elected is L = 1.

inputs			output
A	B	C	L
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

The way to convert this logic table into an equation is again to identify each set of inputs A, B, C that yields an output L = 1. Reading down the table, we find that A =0 and B = 1 and C = 1 yields L = 1. In other words the combination of A' and B and C = A'*B*C = L. But, there are other combinations of A, B, C that yield L = 1 as well. How do we express that? By "OR"ing all such combinations. In this case there are four possible combinations that will get Lois elected.

L = A*B*C' + A*B'*C + A'*B*C + A*B*C

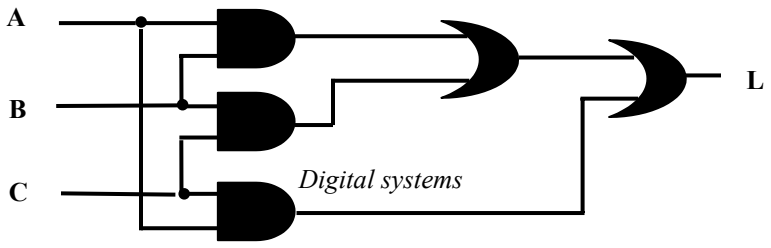
Taking into account the first "OR" rule listed above, we can add two more A*B*C terms and factor to yield

L = A*B*(C' + C) + B*C*(A' + A) + A*C*(B' + B)

Finally,

L = A*B + B*C + A*C .

In other words, Lois is elected if A and B, or B and C, or A and C, vote for her. This can be represented by the following circuit:



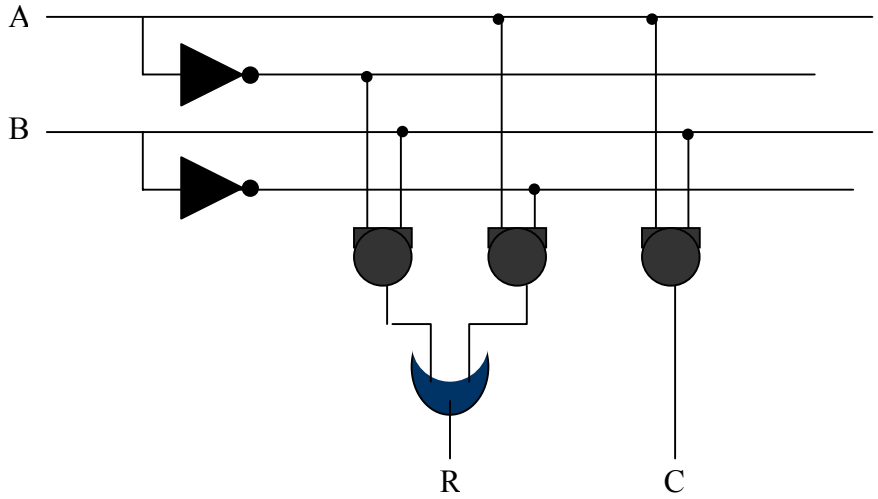
This last example is a little complicated, because we algebraically reduced the equations we obtained directly from the logic table. But that's not absolutely necessary. One can always generate a circuit just from the table itself. Let's do that with the binary adder that we introduced earlier. It's just another logic table. The only feature is that it has two outputs: a result R and a carry C

inputs		outputs	
A	B	R	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Here, as usual, we write down the two equations corresponding to the outputs:

$$A*B = C$$
$$A'*B + A*B' = R$$

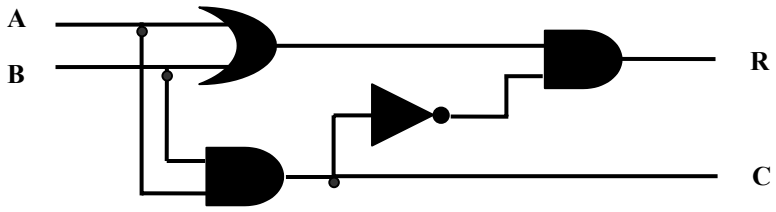
To produce a logic circuit from any logic table, run the input wires and their negatives in one direction, say, horizontally. Then, in a perpendicular direction attach the gates as needed. In this case, we would have:



Now, this is a perfectly good mathematical result. But, it's not a very good engineering result. Notice that it takes six gates to produce the circuit. Since these circuit elements cost money, it's always in our best interest to see if we can improve upon the design, i.e., to see if it can be produced using fewer components. In the voting problem, we algebraically reduced the equations. We can do the same thing here. The equation for the result can be manipulated to become

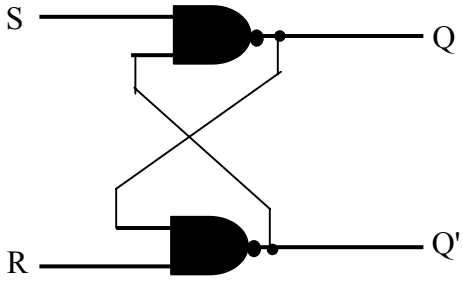
$$(A + B)*(A*B)' = R .$$

Since C = A*B, we can also say that (A+B)*C' = R. So we can produce the following circuit which consists of only four components.



To help you get a feel for these types of circuits, and to give you the opportunity to design your own, try using our virtual laboratory at www.jhu.edu/virtlab/logic/log_cir.htm. It's an interactive program that allows one to create and test logic circuits.

What we have been talking about so far is **combinatorial logic**. In this system, the outputs are directly determined by the inputs. That is, inputs A and B directly determine output C. There is another area called **sequential logic** in which that's not quite the case. In sequential logic, outputs can depend not only on the current inputs, but also the previous inputs. This is accomplished through **feedback**, i.e., using current outputs as inputs to the same circuit. We're not going to go very deeply into sequential logic, because it can get very complicated very quickly, but we should look at one of its simplest forms: the flip-flop or the latch. It's called a "latch" because it's a circuit which can store data. Here's a very simple flip-flop.



Here, we can't establish a logic table because of the feedback loops. But we can analyze what will happen. Suppose S = 1, R = 0. Since R = 0 the output of the lower NAND gate must be a "1", i.e., Q' = 1. Now, since Q' = 1, then the two inputs to the upper NAND gate are "1"s; therefore the output must be "0", i.e., Q = 0.

Schematically,

$$S = 1, R = 0 \Rightarrow Q = 0, Q' = 1.$$

If the two inputs S and R are reversed, then the outputs are reversed because the circuit is symmetric. That is,

$$S = 0, R = 1 \Rightarrow Q = 1, Q' = 0.$$

What if now, we change the inputs from S = 1, R = 0 to S = 1, R = 1. What happens to the outputs? They remain the same. The lower NAND gate's inputs are now "0" and "1", so Q remains "0".

So,

if $S = 1, R = 0 \Rightarrow Q = 0, Q' = 1$
then $S = 1, R = 1 \Rightarrow Q = 0, Q' = 1$

and

if $S = 0, R = 1 \Rightarrow Q = 1, Q' = 0$
then $S = 1, R = 1 \Rightarrow Q = 1, Q' = 0$

In these two cases, the same inputs S = 1, R = 1 yield different outputs. This is completely different from our combinatorial logic circuits.

So how can we use this flip-flop? More importantly, how does it store information? Begin by setting the inputs to S = 1, R = 1. At this point, there's no way to tell what the outputs are. But with a momentary change to S = 0, then back again to S = 1, you know you have Q = 1. That's your data. You can then change or reset the data to Q = 0 by a momentary change to R = 0, then back to R = 1. In either case, as long as R = 1, S = 1, the value of Q will remain. Only when another "0" arrives on either S or R will it potentially change. So we've produced a circuit to store a binary digit of information.

(One peculiarity about this circuit is that, although an input of S = 0, R = 0 is possible, it will produce an predictable output. Try to see why. It depends on the timing of the circuit elements themselves.)

[Note: the Circuit Builder experiment in our Web-based virtual laboratory will not work with sequential logic circuits.]

Enough about logic gates and circuits. The bottom line is that essentially the entire computer industry is based on these primitive elements—just AND, OR, and NOT gates. Of course there's a little more to it than that. . .

We mentioned mechanical relays to serve as logical devices. They can change states—open to closed—maybe 10 times per second. Is that fast enough for today's computing needs? No. Then, in the 1950's vacuum tubes replaced mechanical switches in computers. They were almost 100 times as fast. Still not fast enough. And they generated enormous amounts of heat, and they burned out fairly regularly. Next, in the early 1960's, transistors entered the computer industry. About 10 times faster, yet; lower power consumption; more reliable. These were advances in technology.

Transistors in the 1960's were small cylindrical containers with three connecting wires. But thousands of them were needed for even the simplest of computers. That led to the idea of putting many transistor circuits on a single circuit element or chip—an IC or integrated circuit. Today the Intel Pentium III processor—a single chip—consists of approximately 29 million transistors. And processing speed? Approximately 1 billion cycles per second. Who is creating all these advances? Computer engineers.

Here we should mention the most influential computer engineer of them all: Seymour Cray. His Cray supercomputers have set the standard for compute power for over 25 years. Let's see how his designs have advanced. We'll use "flops" as a measure of power. When you begin to talk about serious compute power, you talk in terms of "flops"—floating-point operations per second. A floating-point operation is one that operates on a decimal number, for example adding 1.23456 to 5.67891. In terms of "flops", here's a brief history of Cray supercomputers:

1976	Cray-1	133 megaflops
1985	Cray2	1.9 gigaflops
1991	Cray C90	16 gigaflops
1996	Cray T3E-900	1.8 teraflops

To obtain these speeds Cray didn't use simply design faster, more complex chips. That's only part of the problem. What about connecting the chips to memory devices or to one another? Have you noticed the size of today's super computers? They're not very big. Do you know why? They can't be any larger. Let me tell you why. First, how far does light travel in a nanosecond—one billionth of a second? About 30 cm; one foot; the length of a sheet of paper. That's not very far. So if you have a computer that can carry out 1 billion operations per second and that can send the result to storage somewhere via a wire which is more than a foot long, then that wire will be carrying two or more pieces of information along its length. In fact, in some super computers information spends as much time in the connecting wires as it does being processed by chips. Pretty amazing.

But there's still more to the problem. Suppose you can solve the space issue. That is, you can develop a new technology to reduce the size of everything, but you still need the same power to drive the system. Now you must dissipate the heat over a much smaller area. And that's not easy. Super-computers often have built-in liquid refrigeration systems to remove the heat from the electronics. When you buy a Pentium III processor, it's quite large. However, the volume is not due to the size of the processor, rather it's due to the heat sink that serves to cool it.

Today's computing is becoming more sophisticated in yet other ways: designing hardware and software so that calculations might be made over multiple processors in the same computer, or designing hardware and software so that calculation might be made over multiple processors in many computers over a network. All the work of computer engineers.

Did you know that $2^{6,972,593} - 1$ is a prime number? It has 2,098,960 digits. That fact that it's a prime was discovered in 1999. How? By linking together over 20,000 computers on the internet in the Great Internet Prime Search (GIMPS). What an idea!

What's the rate at which computer technology is advancing? In 1965 Gordon Moore, co-founder of Intel, observed that, since the advent of the integrated circuit, engineers were able to develop IC's whose transistor density doubled every 18-24 months. That observation became known as Moore's Law. For the past 40 years including today, Moore's Law still seems to be holding. Almost from the beginning soothsayers suggesting that engineers could keep up that rate of improvement; that they were already stretching the limits of technology. But, to date, they have been wrong. So far, engineers have defied these predictions by developing new technologies. And with molecular-level circuitry and storage being explored, who knows how many more years Moore's Law will be extended.

Where is this all going? If Moore's law continues until the year 2023, computers will have the power of the human brain. Impressive. That's computer engineering.

Is there an ultimate computer? Yes. Read about it in the article "The Last Computer" in New Scientist magazine, 02 September, 2000. Will computer engineers be involved? Probably not. This ultimate computer could be a billion-degree laptop or an exploding submicroscopic black hole. I would imagine that an intelligent engineer would avoid such things.