#6: Interlude: Logic Circuits

October 10, 2008

In the previous assignment, I mentioned how Boolean logic is the foundation of modern digital computers. This week, we'll explore what that actually means.

One of the central insights which makes digital computers possible is that it is possible to represent any information you want using only two different symbols—usually referred to as 0 and 1, or F and T. (Sound familiar?) In a computer, these are physically represented by *voltages* (you can think of voltage in a wire as the electrical equivalent of *pressure*, like air pressure or water pressure in a hose). A zero is represented by a voltage of 0V, and a one is represented by some positive voltage (generally around 1.5V nowadays, although it has been getting smaller over time). First we'll talk about how to use these zeros and ones to represent informaion, and then we'll explore how a computer actually computes things with them!

1 Binary numbers

encoding information

So, how is it possible to represent any information you want (such as this document, a recording of Miles Davis's "So What", or that picture of your family at the beach where your little sister isn't looking at the camera and your uncle has his finger up his nose) using only ones and zeros, that is, "true" and "false"? It's obvious that you can't represent any information at all with only *one* symbol—if all you had was zeros, the only thing you could do would be to write down "0000000000000000..."—but it seems incredible that by adding just *one more symbol*, we can now encode anything we want! Adding a third, fourth, or fifth symbol doesn't give us any extra power.

binary

The key is that we can encode any integer we want using only ones and zeros, by writing it in *base two*, also known as *binary*. For this reason, a single zero or one is also called a *b*inary dig*it*, or *bit*. Once we can encode any integer we like, it's not hard to see how to make the jump to encoding images or text files or any other sort of information; we just have to agree on a way to represent the information using numbers.

As you probably recall from elementary school, the system we normally use for writing numbers is *base 10*, which means that the places in a number are

given values based on successive powers of ten. For example, the number "452" has a 2 in the ones (10^0) place, a 5 in the tens (10^1) place, and a 4 in the hundreds (10^2) place, so it represents the number $4\times10^2+5\times10^1+2\times10^0$. The fact that we use a base 10 system also means that we use ten symbols to write numbers: 0, 1, 2, ..., 9.

Likewise, a base two system uses only two symbols (0 and 1, of course) and determines place values based on powers of 2 rather than powers of 10. For example, the number 1101_2 (the subscript 2 indicates that it is the number "one one zero one" in base two, rather than one thousand one hundred one) represents $1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 8 + 4 + 1 = 13_{10}$.

Problem 1. What base ten number does 1001101011_2 represent?

Problem 2. Write 698_{10} in binary.

Problem 3. Compute each of the following. Write your answers in binary.

(a) $0_2 + 0_2$

(b) $0_2 + 1_2$

(c) $1_2 + 0_2$

(d) $1_2 + 1_2$

(e) $1_2 + 1_2 + 1_2$

binary addition

Remember how in elementary school you had to memorize all the "addition facts" (like $3 + 5 = 8$ or $9 + 7 = 16$), and once you had memorized those you could use them to do addition of bigger numbers place-by-place? (For example, if there is a 3 on top of a 5, write "8" underneath; if there is a 9 on top of a 7, write "6" underneath and carry a 1, and so on.) Well, after doing Problem 3, you now know all your binary addition facts! You can now add longer binary numbers in just the same way that you can add longer numbers in base ten.

Problem 4. Add:
$$\begin{array}{rr} & 1011001_2 \\ + & 111100_2 \\ \hline \end{array}$$

Problem 5. Multiply:
$$\begin{array}{rr} & 10010_2 \\ \times & 11_2 \\ \hline \end{array}$$

2 Logic gates

So, now we see how computers can represent data using only ones and zeros. But there's still something missing. How do computers actually *compute* things with ones and zeros? After all, if computers were only good at *storing* data, they would be called "storers," not "computers."

computing with logic gates

The answer is that computers are built out of primitive physical components called *logic gates* which can perform logical operations on the one/zero (true/false) values which are represented as voltages. I won't go into the details of how they actually work—you can look up "logic gates" and "transistors" if you want to read about it. But for now, you can just take my word for it—there are particular configurations of silicon and metal which can transform voltages in a way that implements logical operations.

So, what are some of these logical operations which can be performed? We'll look at four (although there are others as well)—AND, OR, NOT, and XOR. Sound familiar?

AND gate

Figure 1 shows the symbol we use to draw an AND gate. An AND gate takes two voltages as input, and makes its output 1 (high voltage) if both its inputs are 1, and zero (low voltage) otherwise. This should not be surprising—it's exactly the $A \wedge B$ operation that you learned about in a previous week!

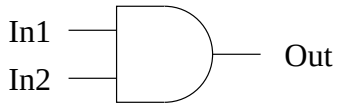

Figure 1: An AND gate

OR gate

Figure 2 shows the symbol used for an OR gate, which implements $A \vee B$. Its output is 0 if both its inputs are 0, and 1 otherwise.

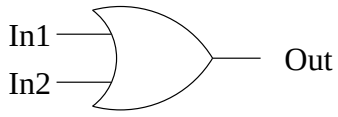

Figure 2: An OR gate

NOT and XOR gates

Figure 3 and Figure 4 show the symbols used for NOT and XOR gates, respectively. A NOT gate simply inverts its input, like $\neg$. An XOR gate implements the "exclusive or" operation $A \oplus B$—its output is 1 if exactly

one of its inputs is 1, and 0 if the inputs are both 0 or both 1. In other words, it implements "not equal to."

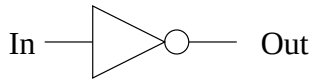

Figure 3: A NOT gate (inverter)

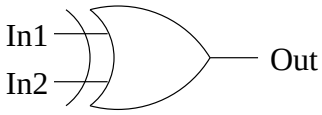

Figure 4: An XOR gate

3 Logic circuits

circuits

The cool thing is that these gates can be wired together to form circuits which compute useful things. Figure 5 shows a simple example.

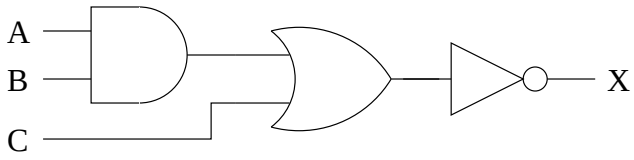

Figure 5: An example circuit

A and B are inputs to an AND gate. The output of the AND gate is an input to an OR gate, along with another input C. Finally, the output of the OR gate is an input to a NOT gate, and the output of the NOT gate is the output of the entire circuit.

Problem 6. Write down an equation defining X in terms of A, B, and C.

Mystery circuit 1

Figure 6 shows another example circuit. By the way, wires only connect at a dot; if two wires cross each other without a fat dot at their intersection (as happens in one place in Figure 6, for example), you should assume that they are not connected and have no effect on each other. (On a physical computer chip, there are several different "layers" for wires to travel through, so you can imagine the wires passing over or under each other in different layers.)

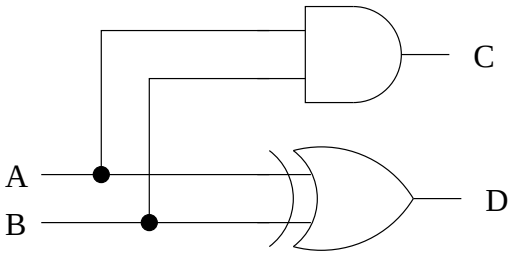

Figure 6: Mystery circuit #1

Problem 7. Fill in the following table showing the operation of the circuit in Figure 6. That is, make a row for each of the possible values of inputs A and B, and show what outputs C and D the circuit computes.

A	B	C	D

Problem 8. Can you describe in words what the circuit in Figure 6 does? (*Hint*: look at Problem 3.)

Problem 9. Figure 7 shows another circuit. You might notice that it contains two copies of mystery circuit #1, and an extra OR gate. Make a table, like the one you made for mystery circuit #1, showing the operation of this circuit. (Hint: the table should have five columns and eight rows.)

Problem 10. Can you describe in words what mystery circuit #2 does?

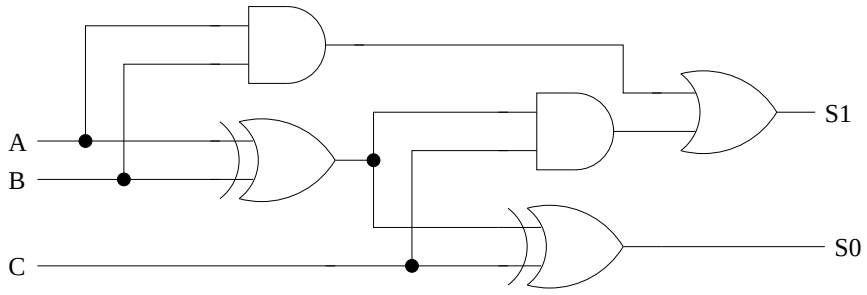

Figure 7: Mystery circuit #2

encapsulating circuits

We'd like to be able to use multiple copies of mystery circuit #2 in bigger

circuits, but drawing out all the individual gates would get tedious, not to mention that it would clutter the diagram and make it hard to understand at a high level. So we can imagine putting a box around the circuit, as shown in Figure 8. Now that we know what mystery circuit #2 does, we can forget about its internal details and just think of it as a "black box" which takes some inputs and somehow computes some outputs. With this perspective, we can use Figure 9 as an abbreviation for the entire circuit. The "M2" in the center stands for "Mystery circuit 2." Figures 8 and 9 show exactly the same circuit, but with different levels of detail.

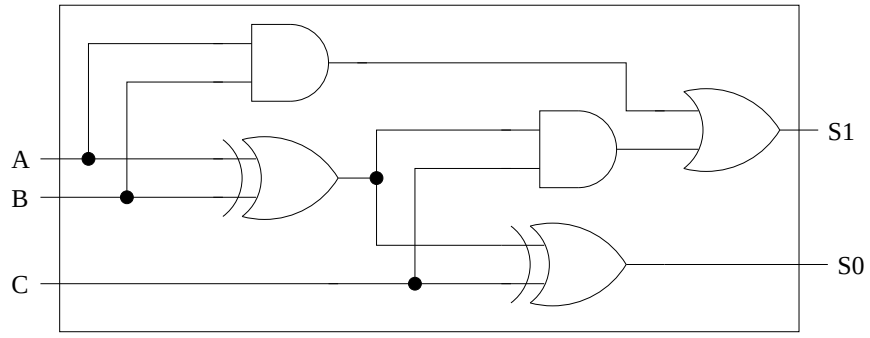

Figure 8: Encapsulating mystery circuit #2

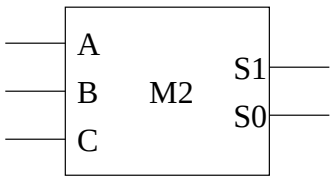

Figure 9: Mystery circuit #2 as a "black box"

Mystery circuit 3

Now that we have a compact symbol for mystery circuit #2, we can use it to draw larger circuit diagrams. Figure 10 shows such a circuit.

Problem 11. In words, what does the circuit in Figure 10 do? (*Hint*: look at Problem 4.)

You may be interested to know that there is a circuit very much like mystery circuit #3 in your computer! It is probably much bigger (featuring perhaps 32 or 64 copies of M2) and much more complicated (in order to make it faster), but the basic idea is still the same.

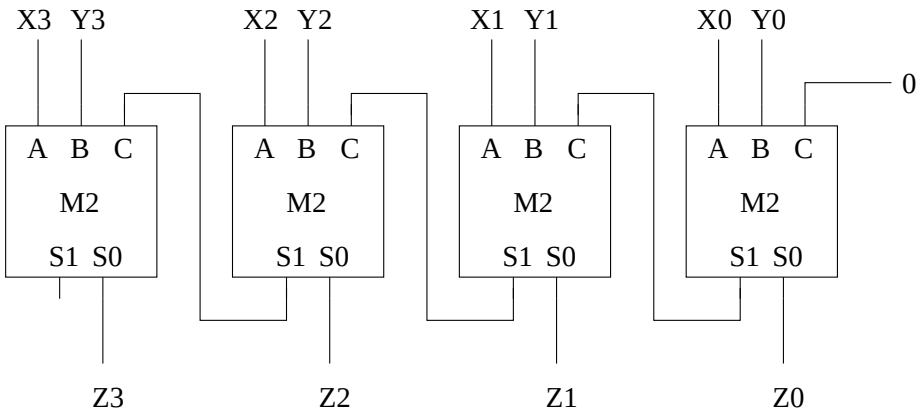

Figure 10: Mystery circuit #3

SR-latch

One final circuit for you to consider: Figure 11 shows a circuit known as an "SR-latch". As you may notice, the strange thing about this circuit is that it takes some of its own outputs as inputs!

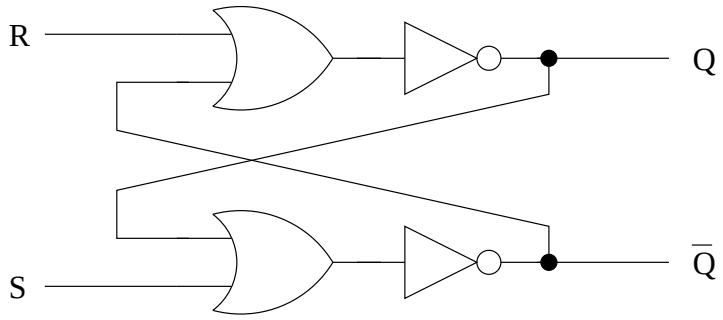

Figure 11: An SR-latch

feedback

Feeding the output of a circuit back into itself as input could cause one of several things to happen. First, it could cause an inconsistent or oscillatory state, where the voltages in the circuit switch back and forth very rapidly between 0 and 1. For example, Figure 12 shows such a circuit: if a 1 is fed into the NOT gate, it outputs a 0, which becomes the new input to the NOT gate, so it outputs a 1, so it outputs a 0, so it outputs a 1... and so on, billions of times per second! Circuits like this are not very useful.

However, sometimes feedback can be useful, and can lead to a consistent, non-oscillating state where the feedback from the output reinforces the input.

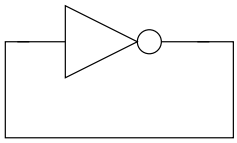

Figure 12: An oscillatory feedback loop

Problem 12. Suppose the SR-latch circuit shown in Figure 11 starts out with $R = S = Q = 0$ and $\overline{Q} = 1$. Is the whole circuit in a consistent state, or will some of the wires oscillate between 0 and 1? Justify your answer.

Problem 13. Now suppose that S changes to 1. What happens to Q and $\overline{Q}$?

Problem 14. Now suppose that after being set to 1, S returns to being 0 again. Now what happens to Q and $\overline{Q}$? (Think carefully about this one!)

Problem 15. What would happen now if R was set to 1?

Problem 16. Why do you think Q and $\overline{Q}$ are called Q and $\overline{Q}$ instead of, say, Q and P?

Problem 17. What could you use an SR-latch for in a computer?

