Laboratory 4

Logic Circuits

Objectives

- Experiment with digital logic circuits using a simple logic gate simulator.

References

Software needed:

1) A web browser (Internet Explorer or Netscape)
2) Applet from the lab website:
   a) LogicGates applet

Textbook reference: Chapter 4, pp. 96–112.

Background

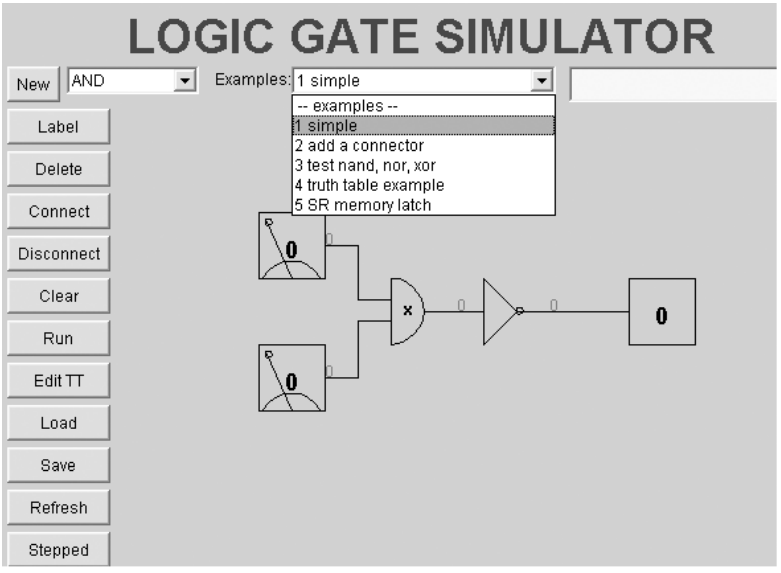
Everything you need to learn is explained in Chapter 4, "Gates and Circuits."

Activity

Digital logic circuits are the bedrock of computer design. CPU chips such as the Intel Pentium or AMD Athlon are silicon wafers in which millions of circuits are embedded. Everything your computer does depends upon them.

In this lab, we will learn how to use the "LogicGates" applet to draw and test digital logic circuits. Chip manufacturers, such as Intel and AMD, use sophisticated (and extremely expensive) software to help them design and test their chips. But the basic idea is the same.

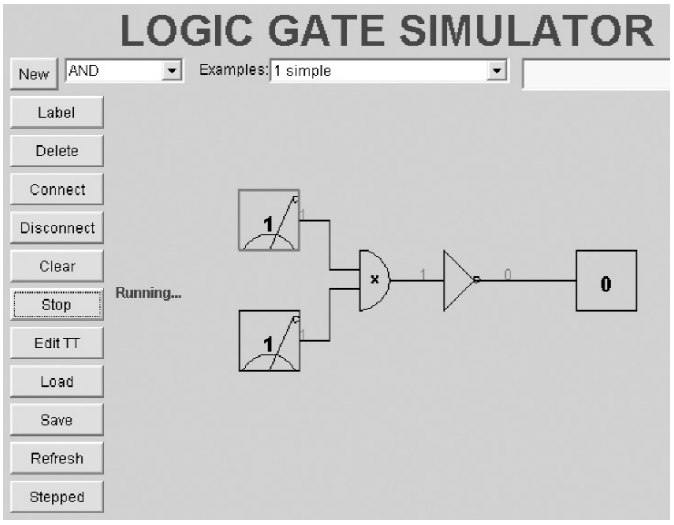
First, start the "LogicGates" applet. You can load a number of examples by pulling down the example choice pull-down menu. Load Example 1 "simple," as shown here:



Each logic gate has a distinctive shape (see pp. 96–100). The wire coming out of the gate shows its current value. Gates are oriented so that their inputs come in from the left side and their outputs go out from the right side. This cannot be changed nor can the gates be rotated in this applet.

Two special boxes exist for input and output. The switch box has a lever in it along with 1 or 0. To flip the lever and change the value, click on the number right in the center. The other box has only 1 or 0 in it. This is the output box that accepts a wire from a gate and displays its value.

To propagate new values through the circuit, click on the *Run* button. It changes to say *Stop* so that when you click on it again, the simulator stops running. When the circuit is running, you can click on the switches and watch values change throughout the circuit.



To practice using the simulator, we'll build the logic circuit shown on p. 105 of the textbook. First, if your simple example circuit is still running, press the Stop button. Now, clear the screen by clicking on the *Clear* button. To build the circuit, you'll need three switches, two AND gates, one OR gate, and one output box. To create them, click on the *New* button, pull down the gate menu and select the gate or other item you want. Then click once where you want the switch, gate, or output box to sit. You can reposition gates, switches, and outputs by dragging them to a new location.

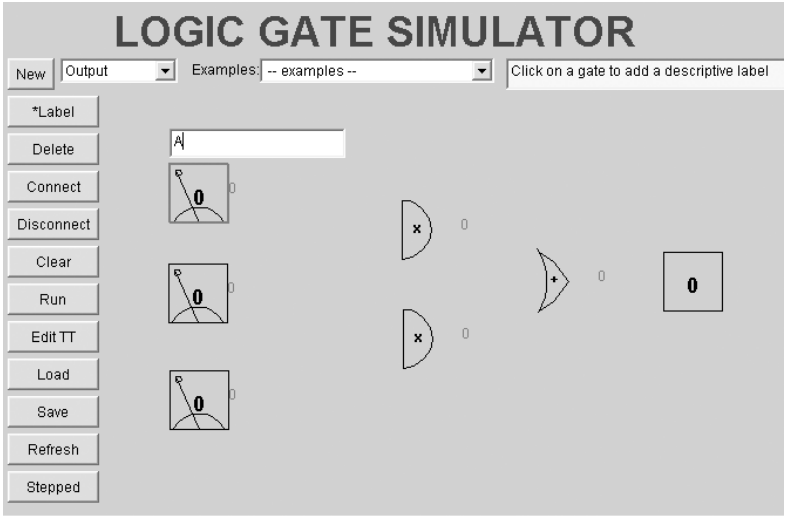
Notice that when you are adding gates, the *New* button changes to say **New*. Most of the buttons in this applet work like this: they show they are still active by displaying an asterisk in front of the label on the button. Because the button stays active, you can click on different gates without having to re-click the *New* button over and over.

Click on the *New* button again to stop adding components. The asterisk will go away.

Let's also attach labels to the boxes to mimic the circuit diagram on p. 105. However, in the Logic Gate simulator, labels attach to boxes, not wires as they do in the textbook.

First click on the *Label* button, which allows us to label gates and switches. Then click on the top switch. A textfield opens up right above the switch. Type "A" into it and press Return.

Here's what you should see:



Label the middle switch as B and the bottom switch as C. Also label the output X. If you want to be complete, label the top AND gate as D and the bottom AND gate as E. When done labeling, click on the *Label* button so that the asterisk doesn't appear.

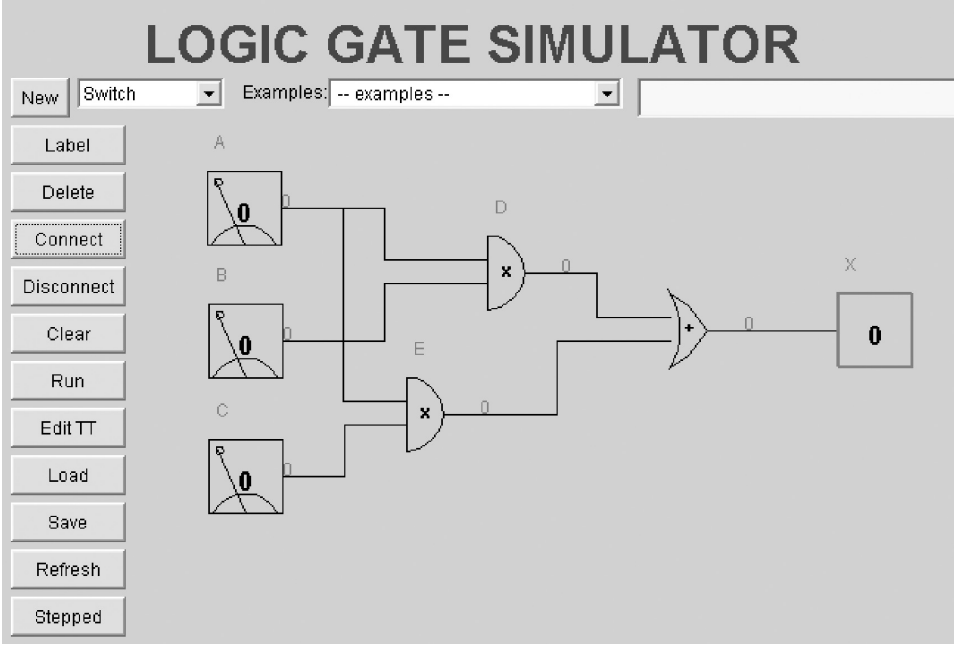
Now it's time to connect the gates, so click on the *Connect* button. Click once on switch "A." A red line flashes into view, anchored in the center of the switch and following the mouse pointer around the screen. Click on the top AND gate. Repeat this process and connect all the boxes so they match the screenshot below. If you make a mistake, click on the *Disconnect* button, then click on a gate. All lines between that gate and all other components will be removed.

A gate's output may have any number of wires coming out of it so that it connects to more than one other gate. Merely select the same gate as the left end of a wire more than once.

Automatic elbows break lines up into horizontal and vertical segments. To see a version that uses diagonal lines, click on the *Stepped* button. However, in most circuit fabrication technologies, all wires must lie on a grid so diagonal wires are either impossible or expensive. To see the elbowed lines again, click on the bottom button, which has changed from *Stepped* to *Diagonal*.

Sometimes the elbows make it hard to see overlapped wires, and that plagues this circuit. So move the bottom AND gate a little to the left until the wires show more clearly.

Here's the final picture:



Now run the circuit by clicking the *Run* button. Click on the switches to change their values. Record the results in a truth table and see if the circuit gives you the same results as shown on p. 106 of your textbook. They had better agree! (They will, don't worry.)

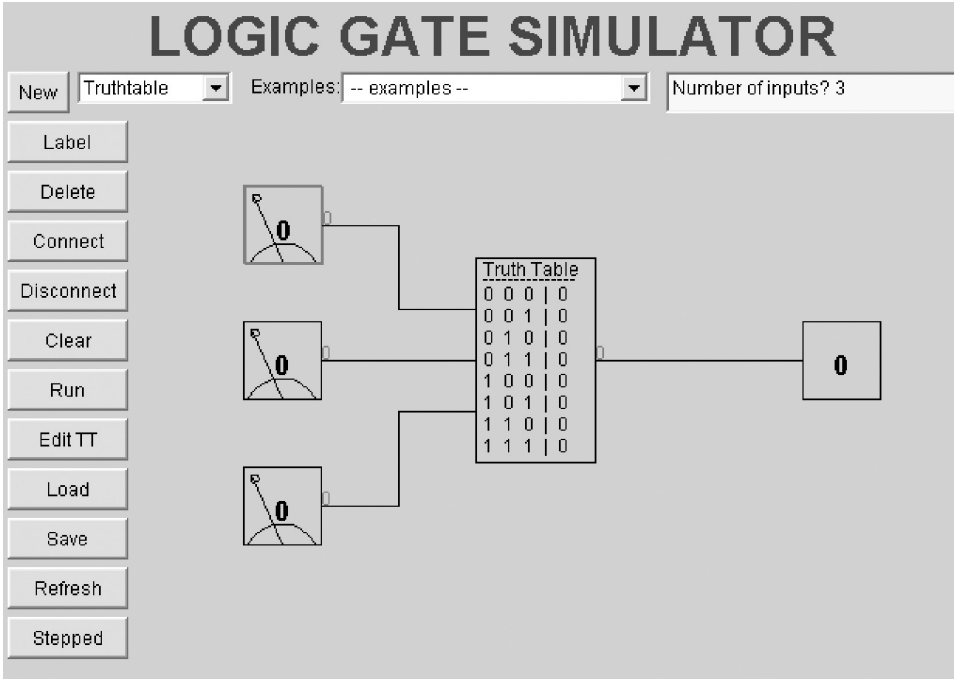
Imagine what a terrible thing it would be if a major chip manufacturer's simulation software didn't give the right values and they sent a chip into production with flaws in it. Well, actually, don't bother imagining: It's already happened! In the early 1990s, Intel recalled their new Pentium processor when scientists and engineers

pointed out that in certain calculations the chip gave the wrong answer! Analysts have estimated that this bug in the chip's division algorithm cost the company about $500 million. (Too bad they weren't using our simulator to check their circuit design!)

Our simulator provides you with six types of gates: AND, OR, NOT, NAND, NOR, and XOR. All of these are defined in Chapter 4 of the textbook. Another type of "gate" available in our circuit simulation is the *Truth table*, which allows you to avoid building really complicated circuits with lots of gates. Instead, you fill in the truth table values and the simulator treats it as a gate. A truth table "gate" can have two, three, or more inputs.

Let's build a circuit using truth tables. Clear the screen and add three new switches and one output.

The trickiest part is putting the right size truth table into the circuit. To do this, make sure the *New* button is activated (has an asterisk next to **New*.) Then pull down on the gates menu and bring *Truthtable* into view. In the yellow message box, Logic Gates will ask you how many inputs your truth table should have. The default is 2, but since we have 3 switches, replace the 2 with 3. Then click once in the middle of the area to create the truth table. Click off the New button and connect these gates as shown below:

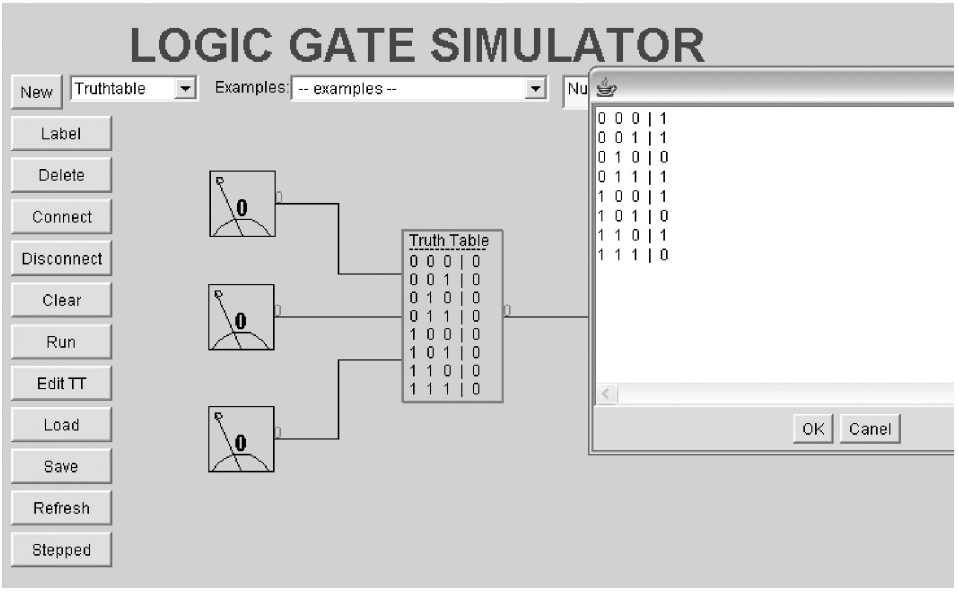


Important note! Make sure that when you connect the switches, you connect the top one first, then the middle one, and finally the bottom one. This is necessary because the order of the incoming wires determines which value is associated with which column in the truth table. The first wire connected will correspond to the first column in the table, the second wire will correspond to the second column, and so on. If you connect them in a different order, the wires going into the truth table will not correspond to the columns of your truth table.

New truth tables list all the input values in canonical order, but since the output is shown as always 0, we need to edit the truth table so it implements our desired output. To edit a truth table, click once on it (after making sure that all the buttons on the left are off, i.e., do not have asterisks in front of their names). This makes the truth table

the top gate and its border will turn red. Now click on *EditTT*. A green window pops up, allowing you to edit this truth table.

The input values are separated from the output value by a vertical bar symbol (|). Go through the truth table and change some of the output values, as shown in the picture below.



Once you are done, click OK to dismiss the truth table editing window, and then click on *Refresh* or just drag the truth table around so that the simulator will repaint the screen.

Run the simulator and play with the switches, comparing the value in the output box to what you see in the truth table. Do they match?

Since the truth table "gate" bypasses digital logic gates altogether and implements the given truth table, it might seem like using it is cheating! Actually, the truth table is similar to SSI chips, shown on p. 113 of your textbook. Sometimes we are more interested in the function that the circuit computes (the pairing up of inputs and outputs) than in exactly which gates are needed to implement that function.

By the way, if there are only two inputs to the truth table, there will be two columns and four rows. If there are three inputs, there will be three columns and eight rows, as shown in the example. Four inputs would require four columns and 16 rows, and five inputs would require five columns and 32 rows. Do you see a pattern?

The input values in all the truth tables you see in this lab are shown in canonical (standard) order, just like the textbook uses. This isn't necessary for the simulator to work properly, but canonical order helps us fallible, distracted humans keep track of our inputs and outputs.

Tip

LogicGates can be used as a standalone Java application. If you use it as an application (not an applet), you can load and save your programs. To run, navigate to the folder containing the LogicGates class files and double click on the **run_application.bat** file.

Exercise 1

Name _____ Date _____

Section _____

1) Start the “LogicGates” applet.
2) Add two switches, one XOR and one output, and connect them.
3) Press the *Run* button and try out all four combinations of inputs for the switches, recording the results in a truth table. Take screenshots for each combination.

Exercise 2

Name _____ Date _____

Section _____

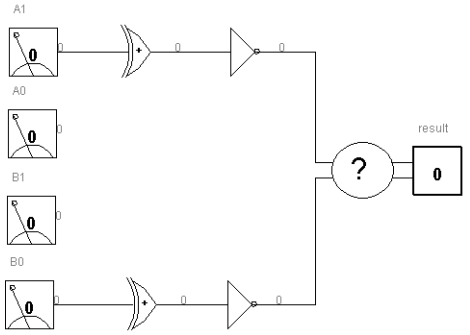
1) Start the “LogicGates” applet.
2) Create the same circuit as previously, but this time insert a NOT box between XOR and the output.
3) Press the *Run* button and try out all four values by changing the switch values. You do not have to take screenshots, but again record the results in a truth table so you can see the values.
4) What does this circuit do? Study your truth table to determine its function. (Hint: It yields a true result only when inputs A and B share a particular relationship. What is that relationship?)

Exercise 3

Name _____ Date _____

Section _____

1) Start the "LogicGates" applet.
2) Build a new circuit that will be a larger version of the one you created in Exercise 2. There will be four switches. Assign them the labels A1, A0, B1, and B0. (Note: A1 is a way of specifying A_1 when you can't really have a subscript.)
3) The circuit will have two XORs, two NOTs, an output, and a mystery box, arranged as shown below:



4) The purpose of this circuit is to compare two 2-bit binary numbers to see if they are the same number. For example, suppose A (that is, the two-digit number comprised of A_1A_0) is 10 and B (the two-digit number B_1B_0) is 10, then the output box will display 1. If A is 10 but B is 11, the output box will show 0, which is Boolean-ese for "false."
5) Connect the XOR boxes to the proper input switches. (Just think about how you compare two numbers for equality. What digits do you compare?)
6) The "mystery gate" is either AND or OR. You can either experiment until you get the right answer, or, better for your brain (and more impressive to your teacher) you can reason out which it should be.
7) Take a screenshot showing your circuit getting the correct result.

Exercise 4

Name _____ Date _____

Section _____

1) Start the “LogicGates” applet.
2) Build a circuit with three switches, two truth tables, and two outputs. Example 4 from the pull-down examples sets up some of this. Choose example 4 and add to it.
3) Set up one of the truth table boxes to implement the Sum column on p. 110 of your textbook, and the other to implement the Carry-out column.
4) Take a screenshot of your program after you run it on one of the combinations of the three inputs.

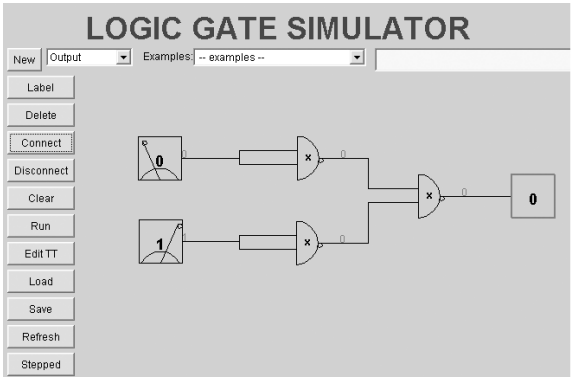
Exercise 5

Name _____ Date _____

Section _____

1) Start the "LogicGates" applet and build the following circuit. It has 2 switches, one output, and three NAND gates.

2) The wiring between the switches and the NAND gates is a little unusual. For both switches, connect the switch to the NAND gate adjacent to it *twice*. This will cause both the wires coming out of the switch to the same NAND gate, though the simulator will spread them out visually. (To do this, click on the *Connect* button, then click once on the top switch, then the top left NAND. Repeat this: click once again on the top switch and then on the top left NAND.)



3) Experiment with the circuit by running it (click on the *Run* button) and change the switch values. Write down the 4 possible inputs and the output you see below:

Top switch	Bottom switch	Output
0	0	_____
0	1	_____
_____	_____	_____
_____	_____	_____

4) What logic gate has the same output?

5) EXTRA CREDIT: Replace the three NAND gates with NOR gates and run the circuit. What logic gate has the same output? Hint: it won't be the same as the answer to number 4.

Exercise 6

Name _____ Date _____

Section _____

A lot of students just do not believe DeMorgan's Law (p. 107 of your textbook) so let's see if the LogicGates simulator can prove it.

1) Start the "LogicGates" applet.
2) Add 2 switches and arrange them vertically on the left side of the panel. The top one we'll call A and the bottom one will be B.
3) We will implement both (AB)' and (A' OR B') in the same circuit. Add two outputs. One output will have the value of (AB)' and the other will have the value of (A' OR B'). If the output boxes read the same for all inputs, then we know that (AB)' = (A' OR B').
4) Since (AB)' is just the NAND gate, add one NAND gate and connect both A and B to it. Then connect the NAND gate to the top output.
5) For the (A' OR B') part, we'll need two NOT gates, connected to A and B. The output of these NOT gates goes into an OR gate, which then goes into the bottom output.
6) Run through all four combinations of binary values of A and B, writing down the two outputs. Are they the same? Is DeMorgan's Law true?

A	B	Top Output	Bottom Output
0	0	_____	_____
0	1	_____	_____
_____	_____	_____	_____
_____	_____	_____	_____

Exercise 7

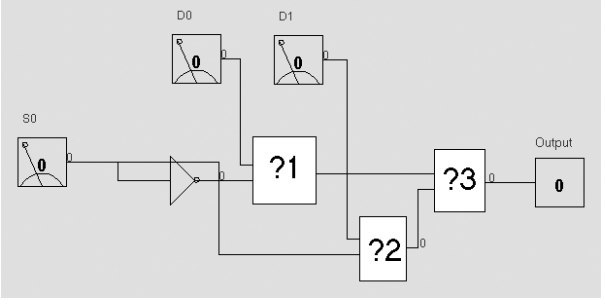
Name _____ Date _____

Section _____

In this exercise we will create a simple multiplexer. Pages 110-111 of your textbook describe how a multiplexer selects which input signal will be copied to the output, based on a set of control lines. If we have 8 incoming signals, we need three select control lines, as shown in the book. Four incoming signals need only 2 select control lines, whereas 2 incoming signals would require only 1. (If you had 16 incoming signals, how many select control lines would you need?)

1) Start the "LogicGates" applet. In this circuit, we will create a multiplexor with two inputs and one control wire.

2) Put three switches in the panel and one output. Label one of the switches S0 and the other two D0 and D1. The S0 switch should be at the left edge and the D0 at the top, similar to the placement on p. 111. You'll need an output at the right edge, as shown below.



3) The tricky part is deciding what the logic gates inside the three mystery boxes should be. All three question mark gates are either AND or OR. You could experiment until you get it right, but let's apply a little brainpower.

   The value in D0, whether it is a 0 or a 1, should be copied to the output box when the select control wire has 0 in it. Similarly, the value of D1 must be copied to the output box when S0 has the value 1 in it. This is why S0 is called a select control input, because it *selects* which of the two Ds gets copied.

   Our task is to figure out which value of S0 will permit D0 through. However, let's tackle the easier case of figuring out which value of S1 will permit D1 through, and then we'll work backward from there.

   When S0 is 1, we want D1. If we *AND* S1 and D1, then the output will be whatever D1 is, whether it is 0 or 1. But if S0 is 0, then we want D0 and not D1. So our AND gate will ignore D1's value and always give us 0.

   By reversing our logic, we see that when S0 is 0 we want D0, so what operation do we apply to them? We can't quite use the same one as above (*AND*) because S0 being 0 would always give us 0. That is why we have to stick a NOT gate in there. Then S0' becomes 1 and if we AND that with D0, we get whatever D0's value is. (Remember that is either 0 or 1. The whole point is to *copy* D0's value through, not force it to be 0 or 1.)

Now we have two cases and two sets of gates: when S0 is 0, select D0's value. When S0 is 1, select D1's value. But we want just one output, not two!

What gate would take 2 or more wires, and "combine" their logic values? You won't get the answer directly here, so think hard about it. If both inputs are 0, the output should be 0. When either input is 1, the output should be 1. Hmmm....

4) Real multiplexors often select between 8, 16, 32, or even more inputs. As your book shows, the binary number represented by the values on $S_2S_1S_0$ permit one of eight data values to flow through to the final F output. Although our logic gate simulator is a little too primitive to permit large diagrams, you might try building one with 4 data inputs and two select control wires. You just might be able to squeeze it all in.

Deeper Investigation

The "LogicGates" applet can create some very complicated circuits. Its main limitation is due to the finite amount of space available—if you have very many gates, you'll eventually run out of room. However, judicious use of the truth table box can cut down on screen clutter.

Example 5 of the sample circuits in the applet is a model of the S-R latch shown on p. 106 of your textbook. It is called memory latch. Try running it. When it starts, the output boxes will cycle between 0 and 1. To stop that, click on one of the switches so that it reads 1. Now the circuit is stable. Experiment with it by setting one of the inputs to 1, then changing it back to 0, and watch the outputs change.

S-R latches are 1-bit memories built out of logic gates. Real computer chips use something like them in the main CPU for registers and other small, high-speed storage. Because the S-R latch has some serious timing problems, variants of it such as the clocked D-latch are used instead, as are even more complicated circuits called *flip-flops*.

S-R latches are circuits that have a peculiar characteristic about their wiring. Can you tell what it is? If not, pretend that the top switch pours a blue-colored liquid through the wires and the bottom switch pours a red-colored liquid. Trace the liquids with colored pens. What do you notice? What would you call this characteristic of S-R latch circuits? Your teacher might be able to give you a hint.

Because of this characteristic, circuits like the S-R latch are called *sequential circuits*. The output of sequential circuits depends not just on the current inputs, but also on the previous values on all the wires. They give a different *sequence* of results over time. Straight-through circuits like the one on p. 105 of your textbook are called *combinational circuits* because their outputs depend only on the combination of their input values.

Try to find out more about logic circuits. Look in some hardware books for an example of a flip-flop circuit. Try tracing its workings.