Important note on scheduling  The second midterm and spring recess give us some obstacles for scheduling the next three assignments. Assignment 5 and Assignment 6 are pushed later than we would like, giving less time for the more substantial Assignment 7. We strongly encourage you to finish Assignment 5 early and get a good start on Assignment 6 before spring recess. That way you can spend most of your time after the break on Assignment 7!

This assignment is about circuits and boolean logic. It will give you an idea about how computers are organized as physical (or electrical) objects. We will be using a tool called Logisim for drawing circuits. You will create a few small circuits which are themselves used as larger components in assembling a whole computer.

Reading  A circuit is assembled from gates, whose symbols appear on the right. Read about circuits in Section 5 of the document *Bits and Logic*.

Also, take some time to read the appendix, later in this assignment. It contains several pointers about using Logisim and has information on where to obtain a copy for yourself.

Submission  There is no template or check file for this assignment. You will submit one file, created by Logisim, named `asgt05.circ`, in the usual way. The file will contain several circuits, one for each problem on the assignment. Please read and follow the instructions later in this assignment for creating and naming individual circuits. Arrange and label your input and output nodes as shown in the block diagrams.

1. Multiplexers and Minterms

Recall the 2-bit decoder example from the document *Bits and Logic*. One can use the decoder pattern to create a multiplexer, a circuit that acts as a switch. The inputs are $2^k$ possible values and $k$ bits that act to select one of the values. The single output is the value specified by the $k$ selection bits. Here is a block diagram of a 4-1 multiplexer.
When the input value \(d_1d_0\) is 01, for example, the signal \(s_{01}\) is passed to the output \(r\).

1. [4 points] Create a circuit named 4-1 multiplexer using only the basic \(\text{and, or, and not}\)-gates. Use the 2-bit decoder for inspiration. Label the input and output nodes as they appear in the block diagram above.

2. [5 points] Use the minterm expansion technique to create a circuit that has four inputs and one output. The output is to be 1 when exactly two of the inputs are 1. Name your circuit exactly two. Begin by writing a truth table with four variables for this problem. You need not submit the truth table.

\[
\begin{array}{cccc|c}
  x_0 & x_1 & x_2 & x_3 & \text{result} \\
  \hline
  0 & 0 & 0 & 0 & 0 \\
  0 & 0 & 0 & 1 & 0 \\
  0 & 0 & 1 & 0 & 0 \\
  0 & 0 & 1 & 1 & 0 \\
  0 & 1 & 0 & 0 & 0 \\
  0 & 1 & 0 & 1 & 0 \\
  0 & 1 & 1 & 0 & 0 \\
  0 & 1 & 1 & 1 & 0 \\
  1 & 0 & 0 & 0 & 0 \\
  1 & 0 & 0 & 1 & 0 \\
  1 & 0 & 1 & 0 & 0 \\
  1 & 0 & 1 & 1 & 0 \\
  1 & 1 & 0 & 0 & 0 \\
  1 & 1 & 0 & 1 & 0 \\
  1 & 1 & 1 & 0 & 0 \\
  1 & 1 & 1 & 1 & 0 \\
\end{array}
\]

II A Circuit from the CS52 Machine

One of the native CS52 Machine instructions is \text{ari}, short for “arithmetic.” It takes three registers, a result and two operands, applies an operation to the operands, and places the result in the specified register. The specific operation is specified by a six-bit auxcode, \(a_5a_4a_3a_2a_1a_0\).

- 0x0: addition
- 0x1: subtraction
- 0x4: bitwise and
- 0x5: bitwise or
- 0x6: bitwise xor
- 0x8: shift logical left
- 0x9: shift logical right
- 0xa: shift arithmetic left
- 0xb: shift arithmetic right

As you can see, only a few of the 64 possible auxcodes are used. In practice, computers will compute many operations simultaneously. They pass all the results to a multiplexer and use the auxcode bits.

There are 16 bits in a CS52 Machine instruction. It takes four bits to specify the opcode, \text{ari} in this case. Each of the three registers takes two bits. That leaves six bits for the auxcode.

Remember that \(2^6 = 64\).
to select the desired result. In the CS52 Machine, the bitwise logical operations are identified by bits $a_5$ through $a_3$ being 0 and bit $a_2$ being 1. Bits $a_1$ and $a_0$ specify the actual logical operation.

The component that carries out the arithmetic instruction is called an arithmetic-logic unit. In this exercise, we will construct a small one-bit unit that handles only the bitwise logical operations. You can imagine combining sixteen more complex units into a device that does all the operations on a full CS52 Machine word.

3. [3 points] Create a circuit named baby ALU that has the structure shown in the block diagram below. The value $r$ will be the result of a bitwise logical operation—and, or, or xor—on the input values $x$ and $y$. Which result appears will be determined by the CS52 Machine’s auxcode bits $a_1$ and $a_0$ from the table above. You may use your multiplexer from Problem 1 and any of the basic gates. Label the input and output nodes as shown in the diagram.

![Block diagram of baby ALU](image)

Sample results:

<table>
<thead>
<tr>
<th>$a_1$</th>
<th>$a_0$</th>
<th>$x$</th>
<th>$y$</th>
<th>$r$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>and</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>or</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>xor</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>xor</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>error</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

III Another CS52 Machine Circuit

Let us turn now to the shifts in the table of CS52 Machine auxcode values. We see that bit $a_0$ determines the direction of the shift (0 for left and 1 for right), and bit $a_1$ determines the nature of the shift (0 for logical and 1 for arithmetic). The only thing that is missing is the magnitude of the shift. For simplicity, we will work with 4-bit words, so the magnitude of a shift can be specified with 2 bits.

4. [3 points] Begin by creating a special kind of multiplexer that is responsible for one bit of a shift. Call your circuit a shiftplexer.
If shift? is 0, then there is no shift; the “bit from here” is passed to the result t. If shift? is 1 and left/right is 0 signifying a left shift, then the “bit from right” is passed to the result. If both shift? and left/right are 1, then the “bit from left” is passed to the result.

5. [5 points] Use your shiftplexer components to construct a 4-bit barrel shifter with the structure shown in the block diagram below. Name it barrel shifter and label the nodes as shown.

The four input bits along the top are the bits to be shifted. The four input values on the left specify the type (logical/arithmetic), direction (left/right), and magnitude (d1d0, interpreted as a 2-bit binary number) of the shift. The bits at the bottom are, of course, the result of the shift.

You will need eight shiftplexers arranged in two rows. One row will have a shift distance of one and will be controlled by d0. The other row will have a shift distance of two and will be controlled by d1. The input bits will flow into the first row of shiftplexers, the the output bits from that row will go into the next row of shiftplexers, and then finally the output bits of the second row are the output bits of the circuit. In addition to the four bits to be shifted, you will need a bit to be shifted in from the right (always 0) and another bit to be shifted in from the left (sometimes 0 and sometimes s11).

Here are a few sample results:
## Hints and observations

- Make sure that you understand the description above. Explain how you can get a shift distance between 0 and 3 positions by creating the two rows of shiftplexers.
- As you do when developing code, take an incremental approach. For example, you might create just one row of shiftplexers and then test the circuit to make sure it works. Then add the next row of shiftplexers. However you develop the circuit, do it in a way that allows you to test as you go; Logisim has good functionality for playing with a working circuit.
Appendix: Working with Logisim

Logisim is installed on the computers in the department's laboratories. Find it under Applications. It is (usually) an easy program to use. There is a tutorial in the Help menu that you can use to get started.

The four main areas of the Logisim window are the small Toolbar in the upper left corner, the Explorer Pane below it, the Attribute Table in the lower left, and the Canvas occupying most of the area on the right. The Canvas is where you will be using your mouse to construct the circuits.

Creating and naming circuits in Logisim  In the main menu, under Project, select Add Circuit. Then type the name of the circuit (as specified in the problem statement) in the window that opens. A new Canvas will open, and its name will show in the Explorer Pane. Do this before you start to create the circuit.

Once you have created a circuit, you can use it as a component in subsequent constructions. Just click on the name in the Explorer Pane and then in the Canvas of the new construction. Any changes you make later to the component circuit will be reflected in all the places where it is used.

Adding components and wires  To add a component, click on its image in the Toolbar and then again in the canvas where you want to the component to be. Once on the canvas, a component can be moved to where you want it. When a component is selected, the Attribute Pane displays several options, like the orientation of the component, the size of the component, the location of the label, the content of the label, and the number of inputs. You will frequently have to change the attributes from their default values.

To add a wire, hold the mouse button down while dragging the mouse. To route wires neatly, create them in segments—either straight lines or single right angles.

Input ports are the square nodes with green dots. Output ports are the round nodes with green dots. If you need multiple copies of a port or another component, it is easiest to create the just one and adjust its properties. Then you can copy-and-paste to make identical copies and move them into place.

Logisim has many different circuit elements that we will not use. Nearly all the gates that you will need appear in the Toolbar. Two exceptions are the constant value (as opposed to a variable input),
which you can find in the Explorer Pane under Wiring, and the xor-gate, which is in the Explorer Pane under Gates.

**Laying out your circuits** You will see, after only a few minutes with Logisim, how easy it is to create a tangled mess of wires and components. To the extent you can, sketch the circuit on paper before trying to construct it in Logisim. That way, you can avoid moving existing components to make space for a new one.

It is usually best to place the components before adding wires. Line them up so that the wires are as straight as possible. Try to make your circuit as regular as possible, for example by aligning similar components, keeping distances between components uniform, and having wires with the same function follow similar paths.

Remember to arrange the nodes along the edges of the circuit, in the same relative positions as they are shown in the block diagrams. For this assignment, input nodes are at the top, control nodes (which are a form of input) are on the left side, and output nodes are at the bottom.

If you move a component and the wires get twisted, just delete the wire segments one at a time and then add them back in. More drastically, if you somehow created a mess, it is often easier to delete the whole thing and start over. At that point, you know how the circuit will look, and you can quickly regenerate the components and connections.

**Testing your circuits** When you have completed a circuit, all the wires should be green. Here is Logisim’s color coding:

- **Dark Green:** The wire is carrying a 0.
- **Bright Green:** The wire is carrying a 1.
- **Red:** The wire has an error value, probably because it is connected to the output of a gate which is unable to determine its value.
- **Blue:** The wire has no value, probably because it is not connected to a source.

You can change the values of the input nodes using the Poke Tool (the pointing finger) and observe the values of the output nodes.

**Solving problems** Sometimes Logisim gets in a bad state. If you find that you’re seeing weird behavior (like input wires with error codes), try saving your file and restarting Logisim. On the other hand, sometimes the problem is in your circuit!

One advantage of working in the virtual world: You don’t create garbage that pollutes the planet. You can discard a circuit and start over with a clean conscience.
Saving your circuits  Select File → Save. The first time you save a circuit, you will be prompted for a file name and a directory. Make sure both are correct.

To return to an existing circuit, open Logisim and select File → Open or File → Open Recent.

Obtaining your own copy of Logisim  Logisim is a Java application that should run on nearly any platform. The program and its documentation can be found at http://www.cburch.com/logisim/.