Lab 7: RPN Calculator

Purpose

The purpose of this lab is:

1. To get familiar with the use of shift registers
2. To design a stack using a customized shift register
3. To design and implement a calculator using the stack and the previously designed ALU

Background Information

1. Reverse Polish Notation (RPN) and Stack Implementation

In this lab you will design a fully functional RPN calculator. RPN is a particular format for representing mathematical expressions. Calculators with RPN were once very popular in the engineering community.

In RPN, an expression is written by putting the operands first, then followed by the operator. For example, “one plus two” in RPN would be written as:

\[
1 \ 2 \ +
\]

An RPN expression with multiple operators is organized in the following way: the operators are read from left to right. The leftmost operator is executed first. It operates on the numbers immediately precede (i.e. to the left of) it. The result of the first operation becomes the operand of the second operation (i.e. the second leftmost operator), and so on. For example, the RPN expression

\[
3 \ 6 \ 4 \ - \ *
\]

would represent:

\[
(6 - 4) \ast 3
\]

For more information, read the Wikipedia page on RPN. You may learn something interesting. For example, do you know what reverse Polish notation is derived from? You guessed it! Polish notation!

Notice that the number which appeared first (3) is used last (in the multiplication). This “first-in, last-out” behavior is similar to that of a stack (see Figure 1). For our purposes, think of a stack as a bidirectional shift register with only one serial input (e.g. LSI) and one output (e.g. Q0). Only the “top” of the stack is visible, that is, through Q0. The input to the stack is the LSI, where data can be “pushed” onto the stack. The output of the stack is the Q0, where data is “popped” out every time the register shifts up.
In an electronic RPN calculator, when an operand is entered, it is pushed to the stack. In the previous example, the stack would contain “4”, “6”, “3”, top-down, after the numbers are entered. When an operator is entered, two operands are popped from the stack, the operation is carried out, and the result is pushed back onto the stack. In this example, after “-” is entered, the stack contains “2” and “3”, where “2” is the result of 6-4. After “*” is entered, the stack contains only “6” on the top, which is the final result of the calculation. The progression of the stack is shown in Figure 2.

With RPN, the execution order of operators is defined by the order they are entered, without using parentheses. A complex expression such as ((1+2)*3+4)*(5+6) is simplified in RPN, as:

\[ 1 \ 2 \ + \ 3 \ * \ 4 \ + \ 5 \ 6 \ + \ * \]

Notice that some operations are commutative in nature, i.e. their left- and right-operands are interchangeable. As a result there can be multiple ways to write an RPN expression. An equivalent expression for the previous example is:

\[ 6 \ 5 \ + \ 4 \ 3 \ 2 \ 1 \ + \ * \ + \ * \]

2. **Problem specification**

You are asked to design an 8-bit RPN calculator in this lab. The calculator has 7 push buttons, labeled “Enter”, “Pop”, “+”, “−”, “×”, “÷”, and “XOR”. It also has 8 slide switches for inputting 8-bit binary operands. It uses four 7-segment displays, in the same way as the ALU lab, for displaying the result. Note that you can now have full 8-bit inputs, unlike the ALU lab where inputs were limited to 4-bits. Consider your ALU design and see if you need to make any changes, such as removing sign-extend modules.

Operands are entered by setting them on the slide switches and pushing the “Enter” button. Each time the button is pressed, the number represented by the slide switches is pushed onto the stack.

Operators are entered by pushing the corresponding “+”, “−”, “×”, “÷” or “XOR” button. Each time the button is pressed, the two operands of the operation are popped.
out, the operation is executed, and the result of the operation is pushed back onto the stack.

The stack has a depth of 4, which should be implemented with a 4-stage shift register. The stack can be cleared by continuously pressing the “Pop” button. Each time the “Pop” button is pressed, the number “0” is shifted up from the bottom of the stack.

Negative numbers are represented in two’s complement as in the ALU lab. The 7-segment displays should always show the sign and magnitude of the number on the top of the stack (which will also be the result of any operation). Of course, only negative numbers need a sign.

A typical sequence of operation with the calculator is like this:

<table>
<thead>
<tr>
<th>Your action</th>
<th>Calculator displays</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  Set slide switches to 2</td>
<td>0 (initial number in the stack)</td>
</tr>
<tr>
<td>2  Press “Enter”</td>
<td>2 (pushed)</td>
</tr>
<tr>
<td>3  Press “Enter” again</td>
<td>2 (pushed again)</td>
</tr>
<tr>
<td>4  Set switches to 1</td>
<td>2 (top)</td>
</tr>
<tr>
<td>5  Press “Enter”</td>
<td>1 (pushed)</td>
</tr>
<tr>
<td>6  Press “+”</td>
<td>3 (1+2 pushed)</td>
</tr>
<tr>
<td>7  Press “*”</td>
<td>6 (3*2 pushed)</td>
</tr>
<tr>
<td>8  Press “Pop”</td>
<td>0 (6 popped)</td>
</tr>
</tbody>
</table>
3. **Circuit Implementation**

A suggested block diagram of the circuit is given below. Your main task will be designing the control unit and the stack. You may reuse modules from previous labs, including the ALU.

![Circuit Diagram](image)

**Figure 3: Overall RPN Calculator Circuit Schematic**

We will implement the stack using a customized bi-directional shift register that contains three 8-bit data inputs: T (top), B (bottom), and L (data to load). It will have 4 modes of operations, selected by the 2-bit S input: 00 (hold – no change), 01 (push), 10 (pop) and 11 (pop-and-load). At each clock cycle, the stack will operate in one of the 4 modes, depending on the user’s input:

<table>
<thead>
<tr>
<th>User Input</th>
<th>Register operation for this clock cycle</th>
<th>Mode Control (S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No input</td>
<td>No change</td>
<td>00</td>
</tr>
<tr>
<td>“Enter”</td>
<td>Shift down, taking input from T</td>
<td>01</td>
</tr>
<tr>
<td>“Pop”</td>
<td>Shift up, taking input from B</td>
<td>10</td>
</tr>
<tr>
<td>“+”, “−”,</td>
<td>Shift up all stages except the top one,</td>
<td>11</td>
</tr>
<tr>
<td>“×”, “÷”,</td>
<td>replace (load) the top stage with input</td>
<td></td>
</tr>
<tr>
<td>“XOR”</td>
<td>from D. Note: this is equivalent to: shift</td>
<td></td>
</tr>
<tr>
<td></td>
<td>up twice, then shift down once, taking</td>
<td></td>
</tr>
<tr>
<td></td>
<td>input from D.</td>
<td></td>
</tr>
</tbody>
</table>
The stack is very similar to a bi-directional shift register with parallel load. The schematic of one stage in the stack is given below (Figure 4).

Figure 4: Schematic for a Shift Register with Bit Depth = 1 (i.e. one stage)

The control unit is a combinatorial circuit that generates control signals (2-bit stack mode-select and 4-bit ALU operation-select) based on which one of the buttons has been pushed. We can assume that only one of the 7 buttons can be pushed at a time. Therefore, the control unit is simply a custom 7-to-6 encoder.
Pre-Lab Questions
(Done individually)

NOTE: We will be checking the prelabs at the beginning of lab. Any prelab not ready by the beginning of your lab section will not receive credit. All prelabs must be done individually, not in groups.

1. Write the following expression in RPN: (1+2)*(3+(4*5)+6)

2. Convert the following RPN expression back to standard math, and determine the result: 4 3 + 2 1 --

3. Should Q0 connect to the A-input or the B-input of the ALU? How about Q1?

4. Duplicate Figure 4 four times. Add interconnections to make a complete 1-bit version of the stack as described in Table 2. Label the data inputs (T, B, D), and the mode control (S1 and S0). This is referred to as a bit-slice of a shift register with depth of 4. You will build this during lab.
**Lab Procedure**

These steps are given only as a recommendation. You are free to come up with your own approach. At this point, you should be comfortable with doing the labs, Xilinx, modular design, etc. For the lab final, you will have to complete an entire lab individually, so make sure both partners know how to do all the parts, including VHDL, schematic entry, simulation, and implementation.

You should explain in your lab report the various steps you took to build your RPN calculator. In addition, for each step, we recommend a certain method to build the module (i.e. schematic or VHDL). This is merely a suggestion, and you may use either as you desire.

1. Start a new project. Make sure all your project settings are correct.
2. Create the one-bit register stage *(Figure 4)* using schematic entry. You may use multiplexers and flip-flops from the Xilinx library. Look at the Xilinx library manual on the ESE171 website for more information about library components.
3. Do a behavioral simulation of the one-bit stage.
4. Create a “bit-slice” of the 8-bit stack using schematic entry. This will contain four instances of your 1-bit register stage (step 2). Add connections and I/O pins to create a 1-bit stack. This would be the “bit-slice” of the 8-bit stack. Connect the bottom input (B) to ground. Try to keep your connections clean. It will help you in debugging.
5. Do a behavioral simulation of the bit-slice. Use a stimulus that shows each of the 4 modes is working.
6. Now, using schematic entry, it’s time to build your 8-bit stack, using eight 1-bit stacks (step 3). Make 8-bit buses for the two data inputs (T and D) and the two outputs (Q0 and Q1). Again, try to keep your connections clean. Draw long buses that run from left to right (or top to bottom) along your entire circuit, and then simply make straight connections between the register inputs and the appropriate bus. You will be much happier in life with a clean circuit.
7. Do a behavioral simulation of the 8-bit stack, just to ensure your connections are correct.
8. Make the control unit. VHDL is recommended, but either is fine.
9. Do a behavioral simulation of the control unit.
10. Build your top-level schematic according to Figure 3. Again, you may use modules from previous labs (such as the ALU and the one-pulse). Connect the Q0 output from the stack directly to an 8-bit output pin, which will go to the 8 LED’s on the peripheral board for debugging purposes (similar to the ALU lab). For the clocks, use the usual bits 14-15 from the CB16CE for the switcher, and simply sysclk for all the registers.
11. Create the user constraints file (i.e. assign the pins). Use the main board buttons for “Enter” and “Pop” and the peripheral board buttons for the operators.
12. Implement and test the design on the board, and then demo to a TA.