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 its 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 among 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: \((6-4)*3\)

Notice that the number which appeared first (e.g. 3) is used last (e.g. in the multiplication). This “first-in, last-out” behavior is similar to that of a stack. For our purposes, think of a stack as a bidirectional shift register with only one serial input (e.g. LSI) and one stage 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. E.g. after “-” is entered, the stack contains “2” and “3”, where “2” is the result of 6-4. After “x” is entered, the stack contains only “6” on the top, which is the final result of the calculation.

With RPN, the execution order of operators is defined by the order they are entered, without using parenthesis. A complex expression such as ((1+2)*3+4)*(5+6) is much 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 a 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 4 7-segment displays, in the same way as the ALU lab, for displaying the result.

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, 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 button is pressed, the number “0” is shifted up from the bottom of the stack.

Negative numbers are represented in two’s complement, which is the same 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.

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</td>
</tr>
<tr>
<td>3. Press “Enter” again</td>
<td>2</td>
</tr>
<tr>
<td>4. Set switches to 1</td>
<td>2</td>
</tr>
<tr>
<td>5. Press “Enter”</td>
<td>1</td>
</tr>
<tr>
<td>6. Press “+”</td>
<td>3</td>
</tr>
</tbody>
</table>
7. Press “*” 6 (result of 2 2 1 + *)
8. Press “Pop” 0

Table 1. Interaction with the calculator.

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 the one-pulse circuit, the ALU, the 7-segment decoder and switcher from previous labs.

![Circuit Diagram]

Figure 1. The overall system

We will implement the stack using a customized bi-directional shift register that contains three 8-bit data inputs: T (top), B (bottom), and D (data to load). It will have 4 modes of operations, selected by the 2-bit S input: 00 (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>“+”, “−”, “×”, “÷”, “XOR”</td>
<td>Shift up all stages except the top one, replace (load) the top stage with input from D. Note: this is equivalent to: shift up twice, then shift down once, taking input from D.</td>
<td>11</td>
</tr>
</tbody>
</table>

Table 2. Function table for the stack
The stack is very similar to a bi-directional shift register with parallel load (textbook Figure 7-11). The schematic of one stage in the stack is given below.

![Schematic of one register stage in the stack](image)

**Figure 2. One register stage in the stack**

The control unit is a combinational 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 could be pushed at a time. Therefore, the control unit is simply a custom 7-to-6 encoder.

**Prelab questions**

Before coming to the lab answer these questions, and have them signed off by a TA at the beginning of the lab.

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

2. What is the result of the following RPN expression: \(4 \ 3 \ + \ 2 \ 1 \ - \ -\)

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

4. Duplicate Fig. 2 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).

**In-lab procedures**

*These steps here are given only as recommendation. You are free to come up with your own approach.*

1. Start a new project with a schematic top-level. Leave this top-level schematic empty for now. Add a new schematic source to the project containing a 1-bit register stage (Fig. 2), and create a symbol for it. There is no need to use a flip-flop with asynchronous reset, since we can always empty the stack by pressing the “Pop” button a few times.
2. Add a new schematic source containing 4 instances of the 1-bit register stages. 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. Create its symbol.
3. Do a behavioral simulation of the bit-slice. Use a stimulus that shows each of the 4
modes is working.
4. Add a new schematic source containing 8 1-bit stacks just created. Make 8-bit buses
for the two data inputs (T and D) and the two outputs (Q0 and Q1). Create its symbol.
5. Add a new source for the control unit. You can use either HDL or schematic entry.
Create its symbol.
6. Do a behavioral simulation of the control unit.
7. In the top-level schematic connect the following: The clock divider, 7 one-pulse
circuits, the control unit, the stack and the ALU. Connect the Q0 output from the
stack directly to an 8-bit output pin, which will go to 8 LED’s for debugging purposes
(similar to the ALU lab). Assign package pin locations; using the main board buttons
for “Enter” and “Pop”, and the extension board buttons for the operators. Implement
and test the design on the board.
8. Complete the top-level schematic with the 7-segment decoder and the display
switcher. Implement and download the design, and demo it to a TA.

Hand-in (at the start of the next lab)

You have to hand in a report that contains the following (See guidelines for reports):
1. Course title, Lab number, Lab title, Your names, and date
2. Brief description of the experiment including the goals and theory
3. Circuit schematics (screenshots, include your names)
4. Waveform simulations (screenshots)
5. Discussion of the results indicating proper function.
6. Conclusion