COMPUTER SYSTEMS I — LAB 3
Memory and a 1-bit counter

1 A starting bit of memory: An S-R latch

To get started on this lab, use our existing chips and their gates to create for yourself an *S-R latch*. You can follow our description from class by using actual push-buttons for the S and R inputs.

Specifically, on the *ETS-7000*, you will notice that, in the lower-left corner of the device, there are two push-buttons, P_A and P_B. Near them, there are four outputs on the board: A, $\bar{A}$, B, and $\bar{B}$. When button P_A is **not** being depressed, then $A = 0$ and $\bar{A} = 1$; when P_A **is** depressed, then $A = 1$ and $\bar{A} = 0$. P_B, B, and $\bar{B}$ behave analogously. Thus, these two buttons can serve as S and R for your latch.

2 Enhancing that bit of memory: A D latch

You will then recall that we observed a limitation with our *S-R latch*. We wanted there to be an input, D, whose value, 0 or 1, would be adopted into the memory element at some moment. Specifically, we wanted Q, the output of the memory element, to take on whatever value of D had when another input, C last has the value 1. That is, when $C = 1$, $Q = D$; when $C = 0$, Q does not change—whatever **that** means!

Build upon your *S-R latch*, turning it into a *D latch* That is, have the memory element adopt of D when $C = 1$, but not when $C = 0$. To perform this task, you will likely need to stop using your push-buttons as S and R, but you will have to use one of them as C.

3 A one-bit counter

Now that you have a memory element that can take on a value only when you press a particular button C, let's build a *1-bit counter*. That is, let's build a circuit that counts from 0 to 1 and then "rolls over" back to 0 again. It's like an odometer with a single, binary digit.

To do so, use your *D latch* to store the *current counter value*. That is, its output, Q, shows the current value of the counter. The input to the *D latch*—D—should be the **next** value of the counter. Thus, if the value of the counter is 0, you should be able to press the button (that is, "tick the clock") and advance the counter to 1. Press the button again, and the counter "rolls over" to 0.

A problem: It's a great idea! But it won't work. Your task is to figure out **why is doesn't work**. When you do, come and talk to me, and we'll figure out what needs to happen to fix it!

4 Building a flip-flop

Congratulations! If you're here, you know what the problem is with your 1-bit counter and your *D latch*. So, build a *D flip-flop* to fix your 1-bit counter. Make it work and **show your working flip-flop** to some kind of authority figure.

5 The 4-bit counter

Once you have completed a working flip-flop, you now need to build a *4-bit counter* that counts from 0 to 15, and then wraps back around to 0 again. Like the 1-bit counter, the rate at which this counter progresses

should be controlled by a *clock* input. Thus, after 16 *clock cycles*, the counter should be back to the value at which it started, having progressed through every value once.

Flip-flop chips: The 74LS273 chip contains 8 1-bit D flip-flops. Examining the chip diagram, we see that this is a 20-pin chip (unlike the 14-pin chips that we've used so far). It still requires power (V_{cc}) and ground (GND) connections. Each of the 8 flip-flops has a data input (D) and a data output (Q).

There are two new pins that have not appeared on previous chips for us. The first is the **clock** (CLK) pin. This pin is connected to the clock input of all of the 8 flip-flops in the chip—that is, this chip behaves as an *8-bit register*. Thus, when you cycle the input on this pin, all of its flip-flops will adopt new values. Note that these flip-flops are *rising edge-triggered*—that is, they adopt a new value as the CLK pin transitions from 0 to 1.

The other new and noteworthy pin on this chip in the **negated clear** ($\overline{CLR}$). When this input to this pin is 1, the flip-flops will behave normally. However, if the input of this pin is set to 0, all of the flip-flops in the chip will have their value immediately reset to 0, irrespective of the CLK input. I suggest connecting this pin to its own pulse switch (see below), allowing you to reset your counter to zero with the press of one button.

5.1 How to do it (roughly)

To make a 4-bit counter, you need a 4-bit register—a role that can be performed by a single 74LS273 chip. The value stored by this register, and thus its output value, is, at any moment, the *current state* of the counter—that is, it will emit the counter's current value. You then need a combinational circuit that can use the register's **output** value as an **input**, and then generate the counter's **next** value. Notice that you have already created this combinational circuit in the form of the *incrementor* from Lab 2.

Show your work before moving onto the final part of the assignment.

6 Changing the sequence

Once you have created your basic counter, you need to create a slightly different counter. Specifically, you need a **3-bit counter** that counts through the following arbitrary sequence of values:

000
011
110
100
010
101

Note that the counter should start at 000, progress through the above values to 101, and then wrap around to 000 again. Also note that the period of this counter is 6—that is, not all 8 possible 3-bit values are used. Every 6 clock cycles, the counter should repeat itself.

Once you have this final counter sequence working, show your work.

This assignment is due Friday, September 30, 11:00 am