Lab 4: Combinational Logic, Storage, and C Pointers & Structs
Due Date: Monday 3/5/2018 11:59PM

This lab covers the material in lectures 9 through 11, with additional C-centric material that will help you in subsequent labs.

If you’re unsure about a result, be sure to explain your thinking. In general, you should show your work. A well-reasoned but wrong answer (for example with a minor error) will receive more points than a wrong answer with no context at all.

Problems (100 points total)
1. Draw a 4 input multiplexer (MUX) using AND and OR gates. There will only be one output!
2. I want to take an 8-bit binary string 11111111 (call it by variable name \( x \) and mask off bits 6 and 7 (set them to zero) How do I use it? Use C syntax.

\[
\begin{align*}
\text{unsigned char } x &= 0xff; \\
\text{unsigned char mask} &= \sim(0x3 \ll 6); \\
x &= mask;
\end{align*}
\]

3. Write 31.125\(_{10}\) in binary converting it like we did in class (fixed, not floating point!)

\[11111.001_2\]

4. What is 01110.0110\(_2\) in decimal?
14.375_{10}

5. Consider the hex string 0x73F02000. If we interpret this as a 32-bit floating point number, what would it be in decimal?

\[ S = 0, \quad E = 231, \quad M = 1.11100000001_2. \quad This \ means \ our \ floating \ point \ number \ is \ \]
\[ (-1)^0 \times 2^{231-127} \times 1.11100000001_2 = 2^{104} \times 1.11100000001_2 = 16 \times (1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{1024}) = \]
\[ 2^{104} \times 1.876_{10} = 3.804 \times 10^{31} \]

6. A 4-2 multiplexer takes 4 bits of data input \( X[0 : 3] \) and uses 1 bit of selector input \( S \) to produce 2 bits of output \( Y[0 : 1] \). If \( S = 0 \), then \( Y_0 = X_0 \) and \( Y_1 = X_1 \); if \( S = 1 \), then \( Y_0 = X_2 \) and \( Y_1 = X_3 \). Implement such a multiplexer using two 2-1 multiplexers.

7. I’m building a prototype for a vending machine. It doesn’t take money yet; If you press the a button it will give you the thing you want. There are 8 rows of items and 4 columns.
Each item is labeled with a letter (denoting the row) and a number (denoting the column). Thus I will have items ranging from A0-A3 all the way to H0-H3. Each item is connected to an actuator (a motor) that pushes the item out to vend it. Design the digital circuitry that will activate the appropriate actuator when pressed. You can assume that the user will only press one button at a time, you can assume that someone pressing the button is equivalent to that button being a digital 1, and you can assume that driving a 1 to the actuator will activate it appropriately. Here’s the hitch: we have to be able to reprogram the thing, such that any button could activate any item slot, so you won’t be able to just tie wires from the buttons to the actuators. **Hint:** You’ll need decoders and memory.

8. Consider the C declarations and code below.

```c
int b[4] = {8, 12, 7, 15};
int u = 20, v = 30, *x = &u, *y, *z;
y = &u;
```
\begin{verbatim}
  z = &b[2];
  ++ *x; // (i.e., *x = *x + 1)
  y = &v;
  --z;
  z[1] = 20;
\end{verbatim}

a) What will \( x \) be after the code runs?

\textit{x will not change, but u will change to 21}

b) What will \( *y \) be?

30

c) Draw out the contents of \( b \) after the code runs.

\{8, 12, 20, 15\}

9. Show how I can construct a byte-addressable 8-bit (address width) memory with only decoders with 2-bit inputs. \textbf{Hint}: You’ll need to use hierarchical decoders. Think carefully about what each bit in your address represents.

The key here is to use a hierarchical organization. We organize our memory into 4 hierarchical units. The top-level unit we’ll call a \textit{chip}. The second-level unit we’ll call a \textit{module}. The third-level unit we’ll call a \textit{array}. Finally, the last level is a byte. We break down the memory address into 4 chunks, each of which is 2 bits. The most significant 2 bits select one of four chips (using a 2:4 decoder). The second most significant bits select a module within one of those chips (again with another 2:4 decoder). The third group of bits selects an array within a module, and finally the last group of two bits selects one of 4 bytes within the array. We can see that with this system, we can store \( 4 \times 4 \times 4 \times 4 \) bytes or 512 bytes. This is actually fairly representative of how we organize memory chips.
Hand-in Instructions

Make sure to put your name on your submission. Submissions without names will be given zero points! For code, this means put a comment at the top of your C file with your name on it.

Physical: If you’re submitting a written copy, hand it to one of the TAs or to the instructor. You can also leave it in the instructor’s mailbox in the CS department office, but make sure to get it time stamped when you do (see the “Submitting Work” section of the syllabus).

Digital: If you would like to submit an electronic copy, note that I will only accept PDF files (no Word docs please). Again, see the “Submitting Work” section of the syllabus. Please do not take a poorly lit picture of your assignment. Your grade will suffer commensurately with our inability to read your work. Once you have a PDF, you should submit it on fourier. You should name your file yourid-lab4.pdf where yourid is the thing in front of the @hawk.iit.edu in your e-mail address.

You can first get your PDF (for example, for me it might be called kh123-lab4.pdf) onto fourier like so:

[me@mylocalmachine]$ scp kh123-lab4.pdf kh123@fourier.cs.iit.edu:

Then you can login to fourier via ssh and submit it:

[kh123@fourier]$ cp kh123-lab4.pdf /home/khale/HANDIN/lab4