CS61CL Final Exam Review Questions Round 1 (Josh Hug, August 6, 2009)

Most (though not all) of these problems are hopefully harder than anything on the final. The goal is to test your understanding as deeply as I could in the time I had to put these questions together. Some aren't actually hard but are there to test your confidence in your understanding. If you get stuck at something, try to break the problem into smaller subproblems. I'll go over all of them at the review session.

Problem 1a: Design a two output, single input finite state machine. The machine is to be a "pulse-width counter", meaning that when the output goes from high to low, the system will output a two bit number which indicates how wide the pulse was. If the pulse was wider than 3 cycles, it should still return 3. It should hold this value as long as the input is still 0. If the input goes to 1, the output should reset to 0 on the next clock cycle.

```
Input  : 01000110001110010111100
Output1: 00000001111001110000011
Output2: 00111100000001111100011
```

1b: How would your design change if the output was supposed to reset to 0 immediately upon seeing a 1, so for example:

```
Input  : 01000110001110010111100
Output1: 00000001110001100000011
Output2: 00111000000001101000011
```

1c: Design a single input single output infinite state machine which outputs a 1 ONLY when a 0 occurs after 2^n ones for ALL n. For example, if we show input on our first line and output on our second line, we get:

```
Input  : 01000110001110010111100
Output : 00100001000000001000010
```

Hint: How many bits of state do we need to keep track of our number of 1's? Maybe it is… infinity? Even if it is, you can still do the problem. Just pick a small n at first, then try it for larger n. You'll see the pattern.

Problem 2a: Given a fixed point unsigned number encoding with 8 bits to the left of the decimal point and 12 bits to the right of the decimal point, how many unique fixed point values can we represent? What is the smallest value? What is the largest?

2b: Suppose we now have a simple floating point scheme with 6 bits (a_0 through a_5), where we interpret our bits as follows: $N = -1^{a_0}\ a_1 a_2 . a_3 \times 2^{a_4 a_5 - 1}$. How many unique numbers can we represent? What is the smallest? What is the largest?

2c: Find the smallest *integer* in IEEE floating point which, when divided by 10 and added to itself, yields itself plus one. In other words, the smallest x such that x/10+x=x+1.

2d: Consider a number written in the floating point format used by humans. Let's say we have 6 significant digits, and the exponent can be between -99 and 99, or in other words:

$$N = d_1 . d_2 d_3 d_4 d_5 d_6 \times 10^{\pm e_1 e_2}$$

Where $d_1 \neq 0$, and all other numbers are from the set of single digit integers {0,1,2,3,4,5,6,7,8,9}. How many unique values can we represent in this format?

2e: Design a circuit (you may use bitwise adders/shifters/subtractors/etc. as components) which adds two IEEE floating point numbers. You may assume both numbers are unsigned, and that we can ignore overflow. What are the two inputs to the adder when we add IEEE Floating Point Numbers corresponding to 1 and 213?

Problem 3a.i: Simplify the following SOP into the simplest possible form:
R'S'T+R'ST'+RT+RS

3a.ii: If we could convert any row of the truth table for 3a.i into a don't care, which would allow us to make the simplest sum of products expression?

3b: Simplify the following SOP into the simplest possible form [Hint: Karnaugh Maps]:
BC'D'+A'BCD'+ABCD'+A'B'C'D+B'CD+AB'C'D+B'D

3c: Simplify the following SOP into simplest form:
A'+AB+AB'+AB'C'D'E'F+B'CDEF'+A'BCD'F'G+AB'+FG+B'CDF'+E'F'G'H'+B'CDFGH+A'BCDEF+BFG'H+D'EF+BF+DEF'+EFGH'+DEFGHJK

3d: Simplify the following SOP into simplest form: A'B'C'+A'B'C+A'BC'+A'BC+AB'C'+AB'C+ABC'+ABC

3e: How many products are present in the optimized SOP expression for the 4 input function whose output is a 1 if and only if there are an odd number of 1's. How many inputs are in each product?

Single Stage MIPS:

In these problems, there are no pipeline stages! None! Don't add them. For these, draw the part of the datapath that changes when you implement the stuff I ask you to implement.

Problem D1: Consider the single stage MIPS implementation given in P&H. Imagine that we want an instruction which writes $a1, $a2, and $a3 to the three words in memory starting at the location in the specified register. How would we change the datapath to implement this instruction assuming we can only write one byte to memory in a cycle [and using only one write port]? [Hint: it will take multiple cycles]

To be more precise, when this special instruction, let's call it sstr. Sstr takes one argument. If we have sstr $t0, for example, then the register transfer language definition would be:

M[$t0]=$a1
M[$t0+4]=$2
M[$t0+8]=$3

(You'd never need this instruction in real life, but hey, it's a review problem)

Problem D2: The single stage processor that you guys implemented for Project 3 has one major qualitative difference from the MIPS ISA, namely that branches don't have a delay slot [in other words, the instruction after a branch isn't executed]. How would you change the datapath for project 3 such that the processor now always executes the instruction immediately after a branch?

In other words, t0 should be incremented by one in the code below, but your project 3 would just skip that instruction. Your job is to change the datapath so that it will execute the instruction after a branch.

```
beq $0, $0, later
addi $t0, $0, 1
later:
addi $t7, $t6, $t3
```

In the next set, I'll have some C stuff, some MIPS programming stuff, pipelining, caches, and virtual memory. Should be up by Sunday morning.

Have fun studying.