CS 14 Spring 2001 — Final exam

Name: _____

You have 3 hours for this exam. **Read carefully** and **answer clearly** each question!

1. (10 points) Given the two functions given below, show how you could implement those functions in a circuit using only an EEPROM. You may assume that the EEPROM holds 2-bit data elements, and that there are 16 such elements that can be addressed using a 4-bit address. You may also assume the ability to program the EEPROM, and should list the values and addresses you would use to program it.

$$Y = \bar{A}BC\bar{D} + A\bar{B}C\bar{D} + \bar{A}\bar{B}\bar{C}\bar{D}$$

$$Z = \bar{A}\bar{B}CD + A\bar{B}\bar{D}$$

2. (10 points) Design a new kind of 1-bit memory element that accepts a new value to be stored on both rising *and* falling clock edges. Draw a circuit diagram and argue that it will be triggered on both clock edges.

3. (10 points) Construct a circuit that will repeatedly emit the following 2-bit pattern:

```
Y0 = 0 1 1 1 ...
Y1 = 1 1 0 1 ...
```

4. You're building the addition component 4-bit ALU. You assume that the numbers being added are integers represented in *two's complement*. It is important that your ALU has an *overflow* output, in the event that overflow occurs in adding two 4-bit numbers and producing a 4-bit result.

   (a) (5 points) **Give an example of overflow.** Show an example of two 4-bit numbers and their addition such that the 4-bit result is incorrect due to overflow.

   (b) (10 points) **Draw an overflow detection circuit.** Specify the rules by which you would detect overflow, and a chain of 4 full 1-bit adders, constructed to use ripple-carry, draw a circuit that would emit high when the addition yielded an overflow, and low otherwise. Be sure to label clearly all relevant inputs and outputs.

5. (10 points) Answer the following questions on instruction sets and assembly programming:

   (a) What is a *pseudoinstruction* and why do they exist for MIPS?

   (b) What does the `jal` instruction do? Why is it necessary for procedure calls?

   (c) What is *register spilling?* When must it be performed?

   (d) Recall that in MIPS, the `add` instruction would cause an interrupt on overflow, and the `addu` would not. Why is it important that `addu` be used for manipulation of `$sp`?

6. (15 points) Answer the following questions on the virtual memory hierarchy:

   (a) How does the memory system determine whether or not a fully associative hardware cache contains a given address?

   (b) How many mappings may exist at one time for each virtual address?

   (c) What memory use behavior is the worst case for LRU? Name a policy that would be better than LRU for this case.

   (d) Explain why fully associative and direct mapped caching are special cases of set associative caching.

   (e) What is the TLB? When is it used?

7. (10 points) Answer the following short questions on the I/O bus and devices:

   (a) How does the processor communicate with a device?

   (b) Compare *polling* and *interrupts*. Give examples for which each is more desirable than the other.

   (c) What happens inside a processor when an interrupt occurs?

8. (15 points) Consider the pipelined datapath and control shown on the attached sheet. Furthermore, assume that you are working with a processor that does **not** perform branch prediction — it simply stalls the pipeline until the branch target is known. Add a *branch hazard control unit* to the diagram. Specifically, show what inputs to this unit dictate whether or not a branch is occuring, and which outputs from this unit are needed to install those bubbles into the pipeline. You may add elements to this diagram (MUXes, etc.) in order to be as specific as possible about the way in which this unit changes the pipeline in the presence of a branching instruction. **Also** describe, over a sequence of cycles, how a branch instruction and the target that it selects would move through the pipeline.

9. (15 points) Consider the multicycle datapath and control shown on the attached sheet. We would like to add support for a new instruction, namely `swpneq`, short for *swap-if-not-equal*. If the contents of the two registers are not equal, then the contents of the two registers are swapped. The instruction takes the following form:

```
swpneq $t2, $t3
```

Note that the diagram attached contains minor modifications to the register file, such that *read register 1* can now be specified by bits 25-21 or bits 20-16 of the instruction, and *write register* can be specified by bits 25-21, 20-16, or 15-11.
Making **no other changes to the multicycles datapath**, describe the steps required to execute this instruction. You should enumerate the steps, and make it clear from where each component used takes its input, and what it does with that input.