MC48 Intra-term Exam 2

Serial #:

This exam is an open-book, open-notes, individual-work exam. Please write your name only on this page. Do not turn the page until instructed, in order that everyone may have the same time. Then, be sure to look at all problems before deciding which one to do first. Some problems are easier than others, so plan your time accordingly. You have 50 minutes to work.

Write the answer to each problem on the page on which that problem appears. You may use the back of that same sheet if you need additional space. You may also request additional paper.

Name: _____

Problem	Page	Possible	Score
1	2	12	
2	4	12	
3	5	12	
Total		36	

1. **[12 Points]** One of the differences between Sun's SPARC architecture and the MIPS architecture we've been studying is that the `lw` instruction on the SPARC can specify the address either as the sum of two registers' contents or the way the MIPS does, as one register's contents plus a constant offset.

   (a) In their paper *An Analysis of MIPS and SPARC Instruction Set Utilization on the SPEC Benchmarks*, Cmelik *et al.* report that on the SPARC, the `gcc` benchmark has 15% of its loads use the register+register version (with neither register being `$0`).

   Assume that the same would be true on the MIPS, if it were modified to have this extra addressing option for `lw` instructions. Put this together with the data in the table on page 248 of the text. Roughly what percentage of `gcc`'s instructions could be eliminated with this architectural modification? Why?

   (b) On the single-cycle datapath below (reproduced from page 295 of the text), indicate any changes needed to support this new variant of the `lw` instruction. Briefly justify your answer.

   (c) The control circuit for the single-cycle datapath is specified by a table on page 296, which is replicated on the next page; fill in the extra row for the new register+register variety of `lw` instruction. If you added any new control signals in the previous part of this problem,

add columns for them to this table and show the values in those new columns in all rows, old as well as new.

Instruction	Reg Dst	ALU Src	Memto Reg	Reg Write	Mem Read	Mem Write	Branch	ALU Op1	ALU Op0
R-format	1	0	0	1	0	0	0	1	0
`lw` (old)	0	1	1	1	1	0	0	0	0
`sw`	X	1	X	0	0	1	0	0	0
`beq`	X	0	X	0	0	0	1	0	1
`lw` (new)									

2. **[12 Points]** The latest extension of the MIPS instruction set architecture, known as MIPS IV, has (among other things) two new instructions called `movn` and `movz`, which stand for "move if not zero" and "move if zero." The way they work is as follows: if you do

```
movn $8, $11, $4
```

then the contents of register 11 are copied into register 8 provided that the value in register 4 is non-zero. If, on the other hand, register 4 contained a 0, then the instruction would do nothing. Similar considerations apply for `movz`, except that this time the copying would happen only if register 4 *did* contain 0; any non-zero value would make the instruction do nothing.

(a) Show how to use the new instructions to put whichever is larger of register 8's value and register 9's value into register 10. (If the values are equal, copy either into register 10.) You may use register 1 as an extra register for temporary use. Do not use any conditional branches.

(b) The code you wrote above should involve the execution of fewer instructions than a version using conditional branches would. However, sometimes rewriting code to use `movn`/`movz` rather than conditional branches doesn't reduce the number of instructions executed. None the less, even if the use of `movn`/`movz` doesn't reduce the number of instructions executed, it can still make the program faster. Explain what this has to do with material we've recently studied.

3. **[12 Points]** A new instruction, `WhereAmI`, is to be added to the MIPS instuction set such that

   ```
   WhereAmI $8
   ```

   would put the instruction's location (the value of PC when the instruction is fetched) into register 8. The register number (here 8) appear in bits 20–16 of the machine language instruction, i.e., the Rt field.

   Show the necessary additions to the multicycle implementation's control finite state machine, which is reproduced below from figure 5.47, page 332. Be sure that the value you store into the register is the address of the `WhereAmI` instruction itself, *not* the address of the instruction after it. Assume (as the book does) that the clock cycle is too short to allow an ALU operation and register write to be done in a single clock cycle.