CMPS 3240 **Name (Print):** ____________________

Fall 2017

Final Exam

12/15/2017

Time Limit: 150 minutes Instructor A. Cruz

This exam contains 9 pages (including this cover page) and 6 problems. Check to see if any pages are missing. Enter all requested information on the top of this page, and put your initials on the top of every page, in case the pages become separated.

You may *not* use your books, notes, or any computer/cell phone/tablet/etc. on this exam.

You are required to show your work on each problem on this exam (except multiple choice). The following rules apply:

- **You are allowed to have one cheat sheet**. You may write on both sides. The paper must be 8.5x11 inches. You must turn in your cheat sheet at the end of the test. It must have your name on it.
- **An ID is required**. You will not be able to turn in the test unless you show a photo ID.
- **Mysterious or unsupported answers will not receive full credit**. A correct answer, unsupported by calculations, explanation, or algebraic work will receive no credit; an incorrect answer supported by substantially correct calculations and explanations might still receive partial credit.
- If you need more space, use the back of the pages; clearly indicate when you have done this.

Do not write in the table to the right.

Problem	Points	Score
1	21	
2	12	
3	14	
4	28	
5	28	
6	15	
Total:	118	

Please circle your major (if applicable). This is for ABET accreditation purposes only and will not affect your grade in any way.

1. Computer Engineering
2. Computer Science
3. Computer Information Systems
4. Information Security
5. Electrical Engineering

1. Consider this code:

```
//Assume *a is properly allocated
for( int i = 2; i < length; i++ )
    a[i] = b[i] * 2;
```

`a` and `b` are the same length. Each element is one byte. The addresses of `a` and `b` are in `$t0` and `$t1` respectively. The values of `i` and `length` are in `$s0` and `$s1` respectively. *Please be complete, do not short-hand.*

(a) (8 points) Give MIPS code that unrolls this operation three times.

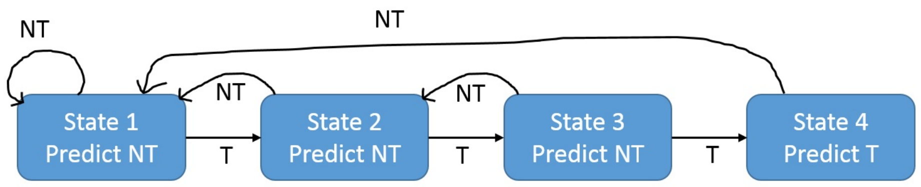
(b) (2 points) Considering part 1, are there restrictions on the length of a and b given the new code?

(Continued from the previous problem)

(c) (8 points) Consider a MIPS processor that has two pipelines. There are no restrictions on what each pipeline can execute. Re-arrange the code into two columns, avoiding hazards and making it as efficient as you can.

(d) (3 points) If `length` is 9, in how many cycles will the code from part finish? How many cycles for part 2? What is the rate of improvement? *Do not use Amdahls law. I do not need to see the entire schedule.*

2. Consider the following branch prediction method:



(a) (2 points) Start in state 1. The processor encounters only Taken branches. How many mistakes will the processor make before it is correct?

(b) (6 points) Consider the following code MIPS pseudo-code:

```
looptop:
...
addi counter, counter, 1
beq counter, limit, looptop
```

The loop body is executed six times before exiting. The processor is in State 1 before arriving at looptop. Profile if a branch is taken or not with the branch prediction method at the top of the page.

(c) (2 points) How many flushes occur?

(d) (2 points) Consider a pre-test loop of the code instead. Would it effect the number of flushes?

3. Consider a direct-mapped cache. The cache has 8 lines/rows, and each row can hold two words. The following addresses are read:

• 4	0b100
• 5	0b101
• 12	0b1100
• 18	0b10010
• 33	0b100001
• 4	0b100
• 26	0b11010

The architecture is word-addressable, and the system is in cold-start.

(a) (7 points) Profile each read. Was it a hit or a miss? Explain.

(b) (7 points) Give the final state of the cache after requesting 26. Include the row, tag and valid bit. *The answer should be a table.*

4. (14 points) Repeat the previous question with a 2-way set associative cache. The cache has 8 rows. Each element can hold only one word.

   (a) (7 points) Profile each read. Was it a hit or a miss? Explain.

   (b) (7 points) Give the final state of the cache after requesting 26. Include the row, tag and valid bit. *The answer should be a table.*

5. (14 points) Repeat question 3 with a fully associative cache. Each row holds two words. The cache has only 4 rows.
   (a) (7 points) Profile each read. Was it a hit or a miss? Explain.

   (b) (7 points) Give the final state of the cache after requesting 26. Include the tag and valid bit. *The answer should be a table.*

6. (15 points) Consider a SIMD instruction set for MIPS, called Alberts Vector eXtensions (AVX). It adds four new integer registers: `$avx0` through `$avx3`, that are quadruple the normal word length.

   AVX has new instructions that segment the large AVX registers four ways to carry out SIMD:

   - `addi.avx <avx register 1>, <avx register 2>, <immediate>`: Which adds the `immediate` value to all four positions in `<avx register 2>` and stores the result in `<avx register 1>`. Similar to x86 intrinsic broadcast.
   - `mult.avx <avx register 1>, <avx register 2>, <avx register 3>`: Multiplies the four values in 2 and 3 and stores the four results in 1.
   - Other ALU operations can end with `.avx` and they will operate as you would expect.
   - `lw.avx <avx register>, offset(<memory>)`: Copies four successive words from `<memory> + offset` into four positions of `<avx register>`. Like with regular `lw`, `offset` must be literal.
   - `sw.avx <avx register>, offset(<memory>)`: Copies the four values in `<avx register>` to four successive positions in `<memory> + offset`. Like with regular `sw`, `offset` must be literal.

   Consider the following code:

```
void IAXPY( ... ) {
    for( int i = 0; i < length; i++ )
        d[i] = a * x[i] + y[i];
}
```

   Convert this code chunk to MIPS.

```
my_IAXPY:
# Assume stack and input validation is taken care of
# Value for a is in $a0
# Memory address for d is in $a1
# Memory address for x is in $a2
# Memory address for y is in $a3
...
```

(Scratch paper)