

CS 573
Midterm Exam
March 13, 2006

The following exam is open book and open notes. You may feel free to use whatever additional reference material you wish, but **no electronic aids** are allowed. Please note the following instructions. There will be a ten point deduction for failure to comply with them:

- start each problem on a new sheet of paper
- write your social security number, but not your name, on each sheet of paper you turn in
- show your work whenever appropriate. There can be no partial credit unless I see how answers were arrived
- be succinct. You may lose points for facts that, while true, are not relevant to the question at hand

You have until 10:20 to finish the exam. The questions are equally weighted.

Here is a little bit of code for a generic three-operand register-based computer, implementing a bubble sort. The questions on this exam are all based on this code. (Note: assume \$0 always contains 0).

```
        addi $1, $0, 40      ; $1 is used to hold a constant 40 to mark end of loop
        addi $2, $0, 0      ; set outer loop counter to 0
outer   addi $3, $0, 0      ; set inner loop counter to 0
inner   ld   $4, awry($3)   ; read awry[i] into $4
        ld   $5, awry+4($3) ; read awry[i+1] into $5 (integers are 4 bytes)
        ble  $4, $5, noswap ; if awry[i] > awry[i+1], swap
        addi $6, $4, 0      ; tmp = awry[i]
        addi $4, $5, 0      ; awry[i] = awry[j]
        addi $5, $6, 0      ; awry[j] = tmp
        st   $4, awry($3)   ; write swapped values back to memory
        st   $5, awry+4($3)
noswap  addi $3, $3, 4      ; increment inner loop counter, see if we're done
        bne  $1, $3, inner
indone  addi $2, $2, 4      ; increment outer loop counter, see if we're done
        bne  $1, $3, outer
```

1. Suppose this code is executed on a computer with an 8K direct-mapped L1 data cache with a 16-byte cache line. Estimate the hit rate in the L1 data cache, assuming the data turns out to have already been sorted when the program is executed. Would a victim cache be likely to improve it?
2. Suppose this code is executed on a processor using a one-bit branch predictor. Estimate the prediction accuracy, assuming all of the elements of the array are in fact already sorted when you begin execution.
3. Qualitatively discuss the extent to which a superscalar, out-of-order implementation is likely to be able to execute this code effectively both with and without register renaming (*i.e.*, how important would register renaming be to taking advantage of the instruction level parallelism in this code?).

4. Consider modifying this code for an EPIC-style processor.

(a) The three lines of code immediately following the `ble` instruction swap two elements of the array, prior to writing the data back out to memory. Show how you can use IA-64 parallel execution semantics (all instructions that execute in parallel read their operands simultaneously, perform the operation, and write their results simultaneously) to do this operation in only two instructions.

(b) Suppose we add predicate registers, and an IA-64 style compare instruction

```
cmp.gt p1 = $10, $11
```

which would compare register \$10 to \$11, and set the predicate bit `p1` to 1 if \$10 is greater than \$11, and 0 otherwise. Use this instruction, and predication, to disable the swap and writeback code instead of using the `ble` from the original code. Of course, you won't be using registers \$10 and \$11, you'll be using other registers.