

CS 573
Midterm Exam
February 26, 2003

The following exam is open book and open notes. You may feel free to use whatever additional reference material you wish, but **no calculators** 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. *Your reasoning is more important than your answer.*
- be succinct. You may lose points for facts that, while true, are not relevant to the question at hand

You have until 12:20 to finish the exam.

1. (30 points) A technique for implementing precise interrupts completely different from any of the approaches described by Smith is to treat the interrupt as if it were a new instruction being inserted into the instruction stream after all of the instructions currently in execution.
 - (a) This technique lets precise interrupts be implemented much more simply than the techniques described in the paper. Why?
 - (b) Unfortunately, it doesn't work for all interrupts. Which of the following interrupts will it work for, and which won't it?
 - i. External interrupts from devices
 - ii. Program interrupts, such as page faults or arithmetic errors
 - iii. Calls to the operating system
2. (15 points) Suppose a simulation is performed on the cache subsystem of a new processor. In analyzing the simulation results, we find that the cache hit rate is unacceptably low. For each of the following circumstances, would we be more likely to improve performance by adding a victim cache, or by making the cache larger?
 - (a) only 39% of the cache blocks contain valid data at the end of the (lengthy) simulation run.
 - (b) 93% of the cache blocks contain valid data almost immediately after the simulation run begins.

3. (40 points) The following code sorts an array:

```
for (i = 0; i < 10; i++)
  for (j = 1; j < 10; j++)
    if (a[j-1] > a[j]) {
      tmp = a[j-1];
      a[j-1] = a[j];
      a[j] = tmp;
    }
```

Assume that the array *a* is already sorted in ascending order when this code is executed.

- (a) When compiled for a reasonably conventional computer (just about anything we've talked about this semester other than IA-64), this code should have three branch instructions, all of them conditional. In words, describe where the three branches are and what they do. How many times is each branch instruction executed (*note: I said executed, not taken*)?
 - (b) If an assume-taken branch prediction strategy is used (Smith's Strategy 1), what will be the prediction accuracy?
 - (c) If Smith's Strategy 2 (if a branch instruction has not been executed before assume it will be taken; if it has been taken before, assume the same thing will happen as on the last execution) is used, what will be the prediction accuracy?
 - (d) If we were to use IA-64 style predicated instructions rather than a conditional branch for one of the three branches in this code, which would it be?
4. (15 points) Alpha's architecture is best described as being a very clean sequential model, while IA-64 is explicitly parallel. Which would be the easier architecture to implement using a superscalar, out-of-order execution model? Or does it matter?