

CS 573
Midterm Exam Solutions
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? *15 points. It doesn't require a reorder buffer or equivalent to make sure instructions are retired in-order; so long as the dependencies are honored (which doesn't require the reorder buffer), it works.*
 - (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
5 points. Yes. There's no need to worry about the ordering of instructions that were already issued before the interrupt arrived; in effect, we just regard the already-issued instructions as already having happened before the interrupt.
 - ii. Program interrupts, such as page faults or arithmetic errors
5 points. No. The problem here is that the interrupt is triggered by an instruction, which is typically partway through execution before it triggers the exception. A later instruction that has already been executed has to be un-executed in this case.
 - iii. Calls to the operating system
5 points. Yes. In this case, it's acting just like a procedure call that also changes to kernel mode.
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*)?

There will be one at the bottom of the outer loop, one at the bottom of the inner loop, and one to branch around the swap code. The exact numbers coming below depend on just what assumptions you want to make about the compiler; if we really make exactly the assumptions above, then there are no unconditional branches. Under those assumptions, the compiler is generating code that assumes the loops are executed once, so the for-loop branches aren't executed the first time the loops are entered, so our numbers will be slightly different than what we came up with in class Friday.

The outer loop branch is executed 10 times.

The inner loop is executed 9 times every time the outer loop is executed. So it is executed a total of 90 times.

The if-branch is executed for every iteration of the inner loop, so it's executed 90 times.

I'm going to be pretty flexible on (1) an extra execution of the loop branches, and (2) assuming you take the branch to do the swap, rather than the other way 'round.

- (b) If an assume-taken branch prediction strategy is used (Smith's Strategy 1), what will be the prediction accuracy?

The outer loop branch is executed 10 times, but taken 9. The inner loop branch is executed 90 times, and taken 80. The if branch is executed 90 times, and taken every time. So we end up with a prediction accuracy of 179/190, or roughly 94%.

- (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?

It will now mispredict the first time through the inner loop, every time but the first. So it'll have 9 new mispredicts. So our accuracy will go to 170/190, or 89%.

- (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?

The only real candidate is the if-code.

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?

I don't expect it would matter. In either case, you're mainly just making reservations; the different instruction semantics will primarily just affect what reservations you're making.