

CS 573
Final Exam
May 7, 2007

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:

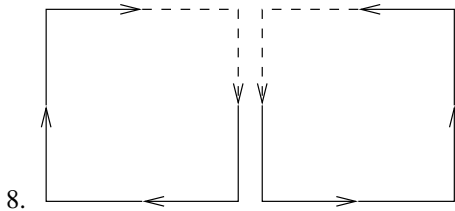
- start each problem on a new sheet of paper
- write your social security number, but not your name, on each sheet of paper you turn in
- show your work whenever appropriate. There can be no partial credit unless I see how you arrived at your answers.
- be succinct. You may lose points for facts that,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

You have until 12:30 to finish the exam.

1. (20 points) Which features of the VAX instruction set make pipelining more difficult? Which features make it easier?
There are two main things that make things more difficult: the variation in instruction sizes (you really can't know where the next instruction starts until you've decoded the current one; you can't even really know where the operands are until you've decoded all the mode bytes). At one time the presence of the really gonzo instructions would have been regarded as an obstacle to pipelining, but sequences to implement them can be put in ROM.
2. The thing that does the most to make it easier is the regularity of the addressing mode formats – but there really isn't much that helps beyond what was mentioned in the midterm: putting all the modes right up next to the opcode rather than interspersing modes and operands (I actually don't know of a machine that is so willfully perverse as that).
3. Note that simply listing features in the hope that they're what I'm looking for didn't get very far. Especially features that aren't part of the ISA (out of order execution, backward compatibility).
4. (20 points) A question that frequently arises when we discuss branch prediction is the feasibility of executing both paths after a branch (rather than just the predicted path), and cancelling the branch not taken.
Thinking about it, it would actually be pretty straightforward to do this for at least a few instructions down both paths, in machines that use the sorts of out-of-order execution we've discussed (papers 08-10).
 - (a) Briefly describe a mechanism for implementing this (you should be able to do this in just a sentence or two).
 - (b) What are at least one advantage and at least one disadvantage of the idea.
5. (15 points) Consider the following two possible organizations for an L1 cache:
 - (a) A direct-mapped split cache (*i.e.* separate I- and D-caches), with each cache holding 8K (for a total of 16K).
 - (b) A 16K, two-way set-associative, combined cache.

What are the advantages and disadvantages of these two organizations?

6. (10 points) Draw a picture like Figure 2 in the Glass and Ni paper (Paper 16) showing which turns are allowed and which are forbidden for a south-first routing algorithm in a two-dimensional grid.
7. In a south-first routing algorithm, you can't turn to go south. So the figure is:



7	Correct for some other direction-first or direction-last routing algorithm not in paper
5	Not a direction-first or direction-last algorithm

10. (20 points) Suppose the following memory transactions are executed on a system using a snoopy cache, using the protocol discussed in the Papamarcos and Patel paper (paper 17).

P1	w(x) 1
P2	r(x) 1

What will the state of the data be in the two processors' caches after this has happened? As usual, assume that neither cache had a copy of the data before the transactions occurred.

- 11. When P1 writes, it acquires x in M (exclusive-modified).
- 12. When P2 reads, it acquires x in S (shared-unmodified). P1 also has it in S at this time.

7	P1 empty; P2 has but didn't say what state.
5	write-through cache; ended with P1 in M and P2 in S
10	P1 in M, P2 in S (but showed first transaction correctly)

14. (15 points) In the second question in HW4, processor P2 executed the statement

$$x = x + 2;$$

On Alewife, this required a RREQ to retrieve x to read, and then a WREQ to be able to write to it. It would have been possible for another processor to have obtained and modified x between the two operations, interfering with the computation.

Suppose we wanted to execute an atomic read-modify-write operation to add two to a shared variable. What message would the processor send the memory location's home to obtain access to the variable to do this?

Just send a WREQ when getting initial access. That'll get the data, and give write access to it.

(a)

5	Only a RREQ