University of California, Berkeley
College of Engineering
Computer Science Division — EECS

Fall 1998 J. D. Kubiatowicz

Quiz 2
November 20, 1998
CS252 Graduate Computer Architecture

You are allowed to use a calculator and one 8.5" x 11" double-sided page of notes. Show your work on all problems. If you find a problem unclear or underspecified, please ask for clarification of the assumptions. If you make assumptions not listed in the problem, please state them clearly in your solutions. Good luck!

Your name:	
SID number:	
Email address:	

1	/30
2	/35
3	/35
Total	**/100**

Question 1: Being Consistent

a) List the three conditions sufficient for sequential consistency that were presented in class. *Hint: one of these dictates the way in which processors can make requests; the other two deal with write atomicity)*

b) Draw a diagram illustrating the conditions of write atomicity for two processors (*hint: this diagram appeared several times in class*)

Question 1 (continued)

c) Describe how the two conditions for write atomicity are satisfied in the following three cases:

1) a snoopy-cache protocol on an atomic bus

2) a snoopy-cache protocol on a split-transaction bus

3) a directory-based protocol in a distributed-memory multiprocessor

Question 1 (continued)

d) Describe how your answer to subpart 3 of part c (the directory-based MP) can lead to livelock in large systems

e) How might you modify the cache controller and/or the directory controller to fix this livelock problem? Give complete architectural details, if possible

f) **EXTRA CREDIT (5 points max):** describe how to exploit the hardware features of a modern out-of-order processor *without* violating sequential consistency

Question 2: Caches and Busses

For this problem, assume that we are dealing with a bus-based shared memory multiprocessor (SMP) using the MESI protocol.

Each processor in the system is identical, and the following information applies to each processor in the system individually. 50% of the instructions executed by each processor are loads or stores (instructions are 32 bits wide). Of these loads and stores, on average 70% are reads to private data, 20% are writes to private data, 8% are reads to shared data, and 2% are writes to shared data.

Each processor has a single-level split instruction/data cache. The instruction cache is 16KB, two-way associative, and has 16 byte lines. The data cache is 16KB, direct mapped, and also has 16 byte lines. The hit rates in the caches are as follows: 97% for private data, 95% for shared data, and 98.5% for instructions. Cache hit time is one cycle for both caches.

The SMP system bus has separate address and data busses, with 64 data lines and 32 address lines. The bus is *atomic* (not split-transaction), so the address and data busses can be treated as a single bus with respect to negotiation. The bus is clocked at one-half the speed of the processor. For reads, memory responds with data 12 bus cycles after being presented the address, and supplies one block of data per bus cycle after that. For writes, both address and data are presented to memory at the same time. Thus a single-word write consumes 1 bus cycle, while a 16-byte write consumes two cycles. Assume all requests are satisfied by the memory system, not by other caches. The processor CPI is 2.0 before considering memory penalties.

a) We want to place as many processors as possible on the bus. What is the bus utilization of a single processor if the caches are write-through with write-allocate strategy? How many of these processors can the bus support before it saturates?

For this part of the problem, assume that a write to the bus automatically invalidates any other existing copies of the data being written. Ignore bus contention and coherence messages received from other processors (*e.g.,* remote snoops), but *do* consider coherence traffic generated by the processor. Show your work.

Question 2 (continued)

b) How many processors can the bus support without saturating if the caches are write-back (and write-allocate)? Assume that the probability of having to replace a dirty block in the cache on a miss that fetches a new block is 0.3. Also assume a MESI protocol, and again ignore bus contention and coherence messages received from other processors (*e.g.,* remote snoops), but *do* consider coherence traffic generated by the processor. Assume that the cache protocol supports upgrades, so a write hit to a shared block causes an invalidate transaction *only*, taking one bus cycle.

Question 2 (continued)

c) Assume we add a unified write-back, write-allocate second-level cache to each processor (assuming a write-through, *no-write-allocate* first level cache with the same parameters as before). The L2 line size is 32 bytes. The local miss rate for *all traffic* to the L2 cache is 10%. The hit time to the L2 cache (which includes the time to transfer data to the L1 cache, but not the L1 hit time) is 6 cycles. The L2 cache is clocked at the same speed as the processor. Inclusion is maintained between the caches. Again, assume that the probability of having to replace a dirty block in the L2 cache on a miss that fetches a new block is 0.3.

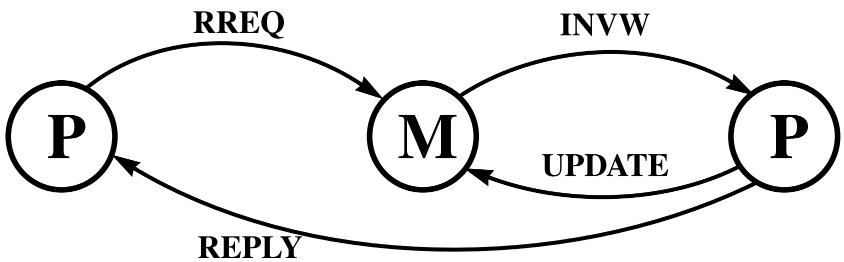
What is the bus utilization of a single processor now? How many of these processors can the bus support without saturating?

Question 2 (continued)

d) Finally, compute the average memory access time (AMAT) in *CPU cycles* for the processor described in part (c). Be sure to include both instruction and data references. First show the equation, then the numerical results.

Question 3: Deadlock

This question examines some of the issues of deadlock that can arise in directory-based distributed memory multiprocessors. For this question, we'll be primarily concerned with the following coherence transaction taken from a DASH-like protocol:



The **P** nodes correspond to processor nodes, and the **M** node corresponds to the home (memory) node for the data in question. **RREQ** is a *read request* message; **INVW** is a *invalidate-writer* message; **UPDATE** is a data reply that updates memory at the home node, and **REPLY** is a data reply to the original requestor.

a) Describe a situation in which the above transaction might occur (that is, what action is the left processor taking, and what is the state of the data in the directory and the rightmost processor).

b) Draw a message dependency diagram (where nodes are messages and arcs correspond to protocol dependencies or queue dependencies). Assume that only one virtual network exists.

Question 3 (continued)

c) This protocol can deadlock. Explain why.

d) One way of removing the deadlocks is by adding a sufficient number of virtual networks. How many virtual networks do you need to remove the possibility for deadlock in this transaction? Why does this remove the deadlock?

e) Given the number of virtual networks above, draw a new version of the message dependence diagram of part (b), labelling the nodes by the virtual network ID that the corresponding messages are sent on.

Question 3 (continued)

f) Assume that, for reasons of cost, you can only have two virtual networks. By introducing a new message type, *proxy*, that can include any information that you wish, describe how to transform this protocol into a purely request-response protocol over the two networks.

g) Draw a new diagram like the *printed* diagram at the beginning of this problem that shows the new use of the proxy message. Label the arcs in the graph with the time-order in which they occur.

Question 3 (continued)

h) Draw a message dependency diagram (as in part b) for these transactions. Be sure to include any queue dependencies. *Hint: this diagram will still include cycles.*

i) Given the fact that the number of outstanding requests from a process is limited, explain how a finite-sized transaction buffer can remove the cycles and the potential for deadlock, yielding a deadlock-free protocol on two networks.