Computer Architecture and Engineering

CS152 Quiz #6

May 7th, 2009

Professor Krste Asanovic

Name:____Answer Key_____

This is a closed book, closed notes exam.

80 Minutes

7 Pages

Notes:

- **Not all questions are of equal difficulty, so look over the entire exam and budget your time carefully.**
- **Please carefully state any assumptions you make.**
- **Please write your name on every page in the quiz.**
- **You must not discuss a quiz's contents with students who have not yet taken the quiz. If you have inadvertently been exposed to the quiz prior to taking it, you must tell the instructor or TA.**
- **You will get no credit for selecting multiple-choice answers without giving explanations.**

Writing name on each sheet	**_____**	**1 Point**
Question 1	**_____**	**25 Points**
Question 2	**_____**	**27 Points**
Question 3	**_____**	**27 Points**
TOTAL	**_____**	**80 Points**

NAME:_____

Problem Q6.1: Sequential Consistency 25 points

Problem Q6.1.A 12 points

Consider two processors with shared memory that are executing the following code. Of the ending conditions shown in the matrix below, which are valid if the system is sequentially consistent? If an ending condition is invalid, explain how sequential consistency would be violated.

Starting assumptions:

- **R** is a register (each processor has its own)
- **X** & **Y** are memory locations
- Initially: **M[X] = 0** and **M[Y] = 10**

Processor A		**Processor B**	
A_1	**Load R,(X)**	B_1	**Store (X),2**
A_2	**Store (Y),11**	B_2	**Load R,(X)**
A_3	**R = R + 1**	B_3	**Store (Y),R**
A_4	**Store (X),R**		

Y \ X	1	2	3
1	SC	Not SC X=2 => A4 < B1 Y=1 => B1 < A4	Not SC X=3 => B1 < A1 Y=1 => A1 < B1
2	SC	SC	SC
3	Not SC Y=3 => B1 < A1 X=1 => A1 < B1	Not SC Y=3 => B1 < A1 X=2 => A4 < B1	SC
11	SC	Not SC Y = 11 => B3 < A2 X = 2 => A4 < B1	SC

NAME:_____

Problem Q6.1.B **13 points**

We want to implement a concurrent stack (LIFO), with possibly multiple users. The code to provide the needed functions (pop & push) is below. You can assume the stack is never empty. The **head** pointer points to the element that is next to be popped off, and when a new element is pushed on, it becomes the head. This code is correct on a sequentially consistent system, but on a system with a fully relaxed memory model it may not be. Insert the minimum number of memory fences to make the code correct on a system with a relaxed memory model. To insert a fence, write the needed fence ($Membar_{LL}$, $Membar_{LS}$, $Membar_{SL,}$ $Membar_{SS}$) in between the lines of code below.

You may assume that the elements put on the stack have two pointers. One points to the data **(->data**) and one points to the next element in the stack **(->next**).

Push(new_element)	**head_element = Pop()**
spin: Test&Set(head_lock, R_{lock})	**spin: Test&Set(head_lock, R_{lock})**
If R_{lock} != 0 goto spin	**If R_{lock} != 0 goto spin**
MembarLL or MembarSL	**MembarLL or MembarSL**
Load R_{head}, (head)	**Load R_{return}, (head)**
Store (new_element->next), R_{head}	**Load R_{next}, (head->next)**
Store (head), new_element	**Store (head), R_{next}**
MembarSS	**MembarSS**
Store (head_lock), 0	**Store (head_lock), 0**
	Return R_{return}

Reference:

```
Test&Set(m, R):
  R ← M[m]
  if R==0 then
    M[m] ← 1
```

NAME:_____

Problem Q6.2: Snooping Cache 27 points

In this problem we will explore designing coherent caches using snooping. To simplify the protocol, the caches have been made write-through with write update. There is no longer a Modified (M) state in the protocol because any time a store occurs, the data is put out on the memory bus and all interested parties (main memory and all caches that contain that block) can update their value. You may assume that the processor is single-cycle and in-order.

To guarantee sequential consistency, when a CPU performs a store, it will stall until its store completes by writing on the bus. As shown in the Figure 1, when a CPU writes, that update goes straight to the bus, and the bus updates the CPU's own cache (as well as main memory and any other copies in other processors' caches).

Since only one component can write to the shared memory bus at a time, a CPU wanting to execute a store may have to wait a large number of cycles. To speed things up, we will add a write buffer. When a CPU performs a store, it is added to the write buffer, and when the bus is available, the store will be broadcast from the write buffer.

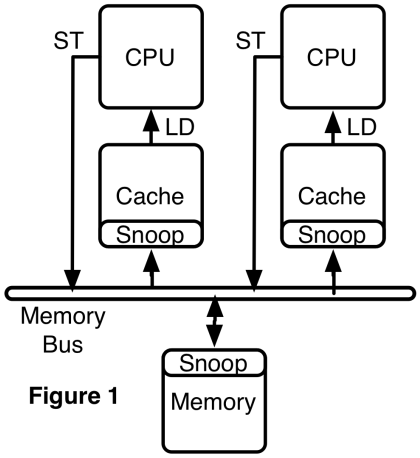

Figure 1

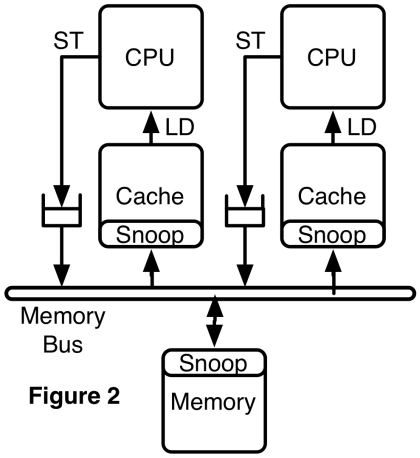

Figure 2

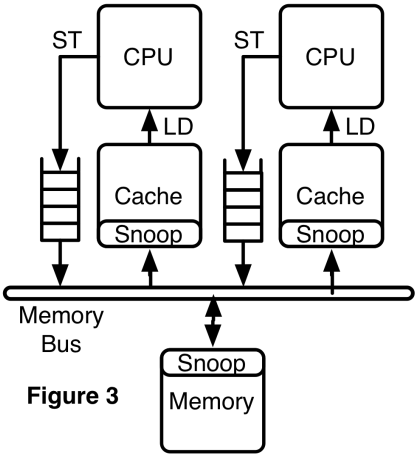

Figure 3

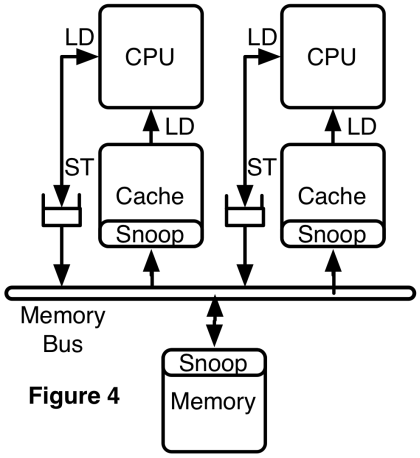

Figure 4

NAME:_____

Problem Q6.2.A **9 points**

Initially the write buffer is only a single entry, as shown in Figure 2. After adding a store to the write buffer, the CPU is allowed to continue execution until it reaches any memory operation (load or store). If it reaches a memory operation while there is still a store in the write buffer, it will stall until it is empty. Is this system sequentially consistent (SC)? Explain. If not, provide a counter-example code sequence and explain how this violates SC.

Yes it is SC. This scheme allows only one memory operation to be in flight at a time and since they are done in order there is no possibility of them being done out of order.

Problem Q6.2.B **9 points**

The write buffer is expanded to have multiple entries (Figure 3). The CPU can now continue execution until it encounters a load, where it will stall until the write buffer is empty. It can continue after stores as long as the write buffer is not full, in which case it stalls. Is this system SC? If so, explain why. If not, provide a counter-example and explain why this is not SC.

Yes this is also SC. Even though multiple stores could be in the buffer at the same time, until the CPU executes a load, it can't differentiate whether the stores have completed or not.

Problem Q6.2.C **9 points**

Finally, the write buffer is reduced back to a single entry (Figure 4), but the CPU is now allowed to continue execution during loads while the store buffer is full (a store must wait for the previous store to clear the write buffer). When performing a load, the CPU will first look for a matching address in the write buffer and forward the data from the write buffer, and if it is not there, it will check its cache as usual. Is this system SC? If so, explain why. If not, provide a counter-example.

No this could violate SC. When the CPU loads from the write buffer, it is seeing a value before other processors can, which breaks SC.

Consider the following example (initially M[X] = 0 and M[Y] = 0):

CPU 0	**CPU1**
ST X,1	ST Y,1
LD Y	LD X

If the stores stay in the buffer, it is possible for both CPUs to load 0's, but that there is no ordering that is SC that would create this.

NAME:_____

Problem Q6.3: Directory 27 points

In this problem we will explore the performance of locks with the coarse directory protocol (Handout #6). Using the **test&set** atomic instruction, a lock can be implemented two ways (Method A and Method B).

Method A (test&set):

```
lock(lk)
   grabbed = 1
   while grabbed == 1
      test&set(lk, grabbed)
```

Method B (test&test&set):

```
lock(lk)
   grabbed = 1
   while grabbed == 1
      if M[lk] == 0
        test&set(lk, grabbed)
```

Test&set(m,R) is implemented in hardware by requesting the line that holds **m** with an **ExReq**. When the CPU gets that line, it checks if **M[m]** is 0, and if so, it sets it. Once the CPU gets the line with **m**, it buffers other coherence protocol requests for the line until the atomic **test&set** has completed.

For this problem, consider two levels of contention:

- **Low Contention:** Only 1 CPU is attempting to acquire the lock, which is currently unlocked but is in the cache of the last processor to acquire the lock.
- **High Contention:** N CPUs are attempting to acquire the lock. On average a CPU will have to retry **r** times. The successful CPU will never be the same as the last CPU to acquire the lock.

With Method B, you can assume that even under high contention, if the **if** succeeds, no other CPU will request the line exclusively during the time interval between the **if** and the **test&set**.

Problem Q6.3.A 9 points

The cache block that holds the lock is currently in a remote cache. When attempting to grab the lock, consider the qualitative number of coherence messages required and their performance implications. Which lock method (A or B) is better under each level of contention? Explain.

i) Low Contention (Method A or Method B better?)

Under low contention Method A will be faster since it only takes one ExReq, instead of a ShReq and a ExReq.

ii) High Contention (Method A or Method B better?)

While many processors are contending for the lock, it is better if they use ShReq because that way many processors can read that the lock is taken without yanking it out of the cache that is using it with a ExReq.

NAME:_____

Problem Q6.3.B **9 points**

The lock is used to provide mutual exclusion for a separate data object. Each CPU will attempt to acquire the lock, and once it does it will: read the data, modify the data, and release the lock. For Method A under each level of contention, is it better for the lock and data object to be in the same cache line or on different cache lines? For this process, consider the qualitative number of coherence messages required and their performance implications. Explain your reasoning.

i) Low Contention (Separate lines better or same line better)

If they are in the same line, it is faster because with one request it can get the lock and the data.

ii) High Contention (Separate lines better or same line better)

Under high contention, its better if caches compete over the lock line, so the CPU with the lock can modify the data without contention.

Problem Q6.3.C **9 points**

If the lock and data are in different cache lines, which locking method is better? For this problem, consider the qualitative number of coherence messages required and their performance implications. Which lock method is better under each level of contention? Explain.

i) Low Contention (Method A or Method B better?)

Like Q6.3.A.i, Method A is faster because it requires only one request to get the lock.

ii) High Contention (Method A or Method B better?)

Like Q6.3.A.ii, under high contention, its better for CPUs to test the lock first with a ShReq.