Computer Architecture and Engineering

CS152 Quiz #6

May 8th, 2008

Professor Krste Asanovic

Name:_____

This is a closed book, closed notes exam.

80 Minutes

10 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 if the instructions ask you to explain your choice.**

Writing name on each sheet	**_____**	**1 Point**
Question 1	**_____**	**32 Points**
Question 2	**_____**	**29 Points**
Question 3	**_____**	**18 Points**
TOTAL	**_____**	**80 Points**

Problem Q6.1: Implementing Directories 32 POINTS

Ben Bitdiddle is implementing a directory-based cache coherence invalidate protocol for a 64-processor system. He first builds a smaller prototype with only 4 processors to test out the cache coherence protocol described in Handout #6. To implement the list of sharers, **S**, kept by **home**, he maintains a bit vector per cache block to keep track of all the sharers. The bit vector has one bit corresponding to each processor in the system. The bit is set to one if the processor is caching a shared copy of the block, and zero if the processor does not have a copy of the block. For example, if Processors 0 and 3 are caching a shared copy of some data, the corresponding bit vector would be `1001.`

Problem Q6.1.A 4 POINTS

The bit vector worked well for the 4-processor prototype, but when building the actual 64-processor system, Ben discovered that he did not have enough hardware resources. Assume each **cache block is 32 bytes**. What is the overhead of maintaining the sharing bit vector for a 4-processor system, as a **fraction of data storage bits**? What is the overhead for a 64-processor system, as a **fraction of data storage bits**?

Overhead for a 4-processor system: _____

Overhead for a 64-processor system: _____

Problem Q6.1.B **8 POINTS**

Since Ben does not have the resources to keep track of all potential sharers in the 64-processor system, he decides to limit **S** to keep track of only 1 processor using its 6-bit ID as shown in Figure Q6.1-A (**single-sharer scheme**). When there is a load (`ShReq`) request to a shared cache block, Ben invalidates the existing sharer to make room for the new sharer (home sends an `InvReq` to the existing sharer, the existing sharer sends an `InvRep` to home, home replaces the exiting sharer's ID with the new sharer's ID and sends a `ShRep` to the new sharer).

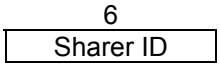

Figure Q6.1-A

Consider a 64-processor system. To determine the efficiency of a full bit-vector scheme and single-sharer scheme, **fill in the number of invalidate-requests** that are generated by the protocols for each step in the following two sequences of events. Assume cache block **B** is uncached initially (R(dir) & dir= ε)) for both sequences.

Sequence 1	**Full bit-vector scheme # of invalidate-requests**	**single-sharer scheme # of invalidate-requests**
Processor #0 reads **B**	0	0
Processor #1 reads **B**		
Processor #0 reads **B**		

Sequence 2	**Full bit-vector scheme # of invalidate-requests**	**single-sharer scheme # of invalidate-requests**
Processor #0 reads **B**	0	0
Processor #1 reads **B**		
Processor #2 writes **B**		

Problem Q6.1.C **8 POINTS**

Ben thinks that he can improve his original scheme by adding an extra "**global bit**" to **S** as shown in Figure Q6.1-B (**global-bit scheme**). The global bit is set when there is more than 1 processor sharing the data, and zero otherwise.

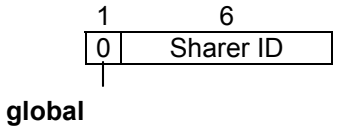

Figure Q6.1-B

When the global bit is set, home stops keeping track of a specific sharer and assumes that all processors are potential sharers.

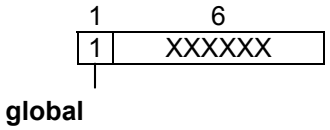

Figure Q6.1-C

Consider a 64-processor system. To determine the efficiency of the global-bit scheme, **fill in the number of invalidate-requests** that are generated for each step in the following two sequences of events. Assume cache block **B** is uncached initially (i.e., R(dir) & (dir = ε)) for both sequences.

Sequence 1	**global-bit scheme # of invalidate-requests**
Processor #0 reads **B**	0
Processor #1 reads **B**	
Processor #0 reads **B**	

Sequence 2	**global-bit scheme # of invalidate-requests**
Processor #0 reads **B**	0
Processor #1 reads **B**	
Processor #2 writes **B**	

Problem Q6.1.D **12 POINTS**

Ben decides to modify the protocol from Handout #6 for his global-bit scheme (Problem 4.4.C) for a **64-processor system**. Your job is to complete the following table (Table Q6.1-1) for him.

Use the same assumptions for the interconnection network, cache states, home directory states, and protocol messages as Handout #6. However, *R(dir)* (if $dir \neq \varepsilon$) now means that there is only one processor sharing the cache data (global bit is unset), and *R(all)* means the global bit is set.

Use *k* to represent the site that issued the received message. For *Tr(dir)* and *Tw(id)* states, use *j* to represent the site that issued the original protocol request (ShReq/ExReq).

No.	Current State	Message Received	Next State	Action
1	R(dir) & (dir = ε)	ShReq	R({k})	ShRep->k
2	R(dir) & (dir = ε)	ExReq	W(k)	ExRep->k
3	R(dir) & (dir ≠ ε)	ShReq		
4	R(all)	ShReq		
5	R(dir) & (dir ≠ ε)	ExReq		
6	R(all)	ExReq		
7	W(id)	ShReq	Tw(id)	WbReq->id
8	Tr(dir) & (id ∈ dir)	InvRep	Tr(dir - {id})	nothing
9	Tr(dir) & (dir = {k})	InvRep	W(*j*)	ExRep->j
10	Tw(id)	FlushRep		

Table Q6.1-1: Partial List of Home Directory State Transitions

Problem Q6.2: Snoopy Cache Coherent Shared Memory 29 POINTS

This problem improves the snoopy cache coherence protocol presented in Handout #7. As a **review** of that protocol:

> When multiple shared copies of a *modified* data block exist, one of the caches *owns* the current copy of the data block instead of the memory (the owner has the data block in the OS state). When another cache tries to retrieve the data block from memory, the owner uses *cache to cache intervention* (CCI) to supply the data block. CCI provides a faster response relative to memory and reduces the memory bandwidth demands. However, when multiple shared copies of a *clean* data block exist, there is no owner and CCI is *not* used when another cache tries to retrieve the data block from memory.

To enable the use of CCI when multiple shared copies of a *clean* data block exist, we introduce a new cache data block state: *Clean owned shared* (COS). This state can only be entered from the clean exclusive (CE) state. The state transition from CE to COS is summarized as follows:

initial state	other cached	ops	actions by this cache	final state
cleanExclusive (CE)	no	**CR**	**CCI**	**COS**

There is no change in cache bus transactions but a slight modification of cache data block states. Here is a summary of the possible cache data block states (**differences from problem set highlighted in bold**):

- Invalid (I): Block is not present in the cache.
- Clean exclusive (**CE**): The cached data is consistent with memory, and no other cache has it. **This cache is responsible for supplying this data instead of memory when other caches request copies of this data.**
- Owned exclusive (OE): The cached data is different from memory, and no other cache has it. This cache is responsible for supplying this data instead of memory when other caches request copies of this data.
- Clean shared (CS): The data has not been modified by the corresponding CPU since cached. Multiple CS copies and at most one OS copy of the same data could exist.
- Owned shared (OS): The data is different from memory. Other CS copies of the same data could exist. This cache is responsible for supplying this data instead of memory when other caches request copies of this data. (Note, this state can only be entered from the OE state.)
- **Clean owned shared (COS): The cached data is consistent with memory. Other CS copies of the same data could exist. This cache is responsible for supplying this data instead of memory when other caches request copies of this data. (Note, this state can only be entered from the CE state.)**

Problem Q6.2.A **16 POINTS**

Fill out the state transition table for the new COS state:

initial state	other cached	ops	actions by this cache	final state
COS	yes	none	none	**COS**
		CPU read		
		CPU write		
		replace		
		CR		
		CRI		
		CI		
		WR		
		CWI		

Problem Q6.2.B **8 POINTS**

The COS protocol is not ideal. Complete the following table to show an example sequence of events in which multiple shared copies of a clean data block (*block B*) exist, but CCI is *not* used when another cache (*cache 4*) tries to retrieve the data block from memory.

cache transaction	source for data	state for data block B			
		cache 1	cache 2	cache 3	cache 4
0. *initial state*	—	**I**	**I**	**I**	**I**
1. cache 1 reads data block B	**memory**	**CE**	**I**	**I**	**I**
2. cache 2 reads data block B	**CCI**	**COS**	**CS**	**I**	**I**
3. cache 3 reads data block B	**CCI**	**COS**	**CS**	**CS**	**I**
4.					
5.					

Problem Q6.2.C **5 POINTS**

As an alternative protocol, we could eliminate the CE state entirely, and transition directly from I to COS when the CPU does a read and the data block is not in any other cache. This modified protocol would provide the same CCI benefits as the original COS protocol, but its performance would be worse. **Explain the advantage of having the CE state.** You should not need more than one sentence.

Problem Q6.3: Relaxed Memory Models 18 POINTS

Consider a system which uses Weak Ordering, meaning that a read or a write may complete before a read or a write that is earlier in program order if they are to different addresses and there are no data dependencies.

Our processor has four fine-grained memory barrier instructions:

- **MEMBAR$_{RR}$** guarantees that all read operations initiated before the MEMBAR$_{RR}$ will be seen before any read operation initiated after it.
- **MEMBAR$_{RW}$** guarantees that all read operations initiated before the MEMBAR$_{RW}$ will be seen before any write operation initiated after it.
- **MEMBAR$_{WR}$** guarantees that all write operations initiated before the MEMBAR$_{WR}$ will be seen before any read operation initiated after it.
- **MEMBAR$_{WW}$** guarantees that all write operations initiated before the MEMBAR$_{WW}$ will be seen before any write operation initiated after it.

We will study the interaction between two processes on different processors on such a system:

P1	P2
`P1.1: LW R2, 0(R8)`	`P2.1: LW R4, 0(R9)`
`P1.2: SW R2, 0(R9)`	`P2.2: SW R5, 0(R8)`
`P1.3: LW R3, 0(R8)`	`P2.3: SW R4, 0(R8)`

We begin with following values in registers and memory (same for both processes):

register/memory	Contents
R2	0
R3	0
R4	0
R5	8
R8	0x01234568
R9	0x89abcdec
M[R8]	6
M[R9]	7

After both processes have executed, is it possible to have the following machine state? Please circle the correct answer. If you circle **Yes**, please provide sequence of instructions that lead to the desired result (one sequence is sufficient if several exist). If you circle **No**, please explain which ordering constraint prevents the result. You will get no credit for circling an answer with no explanation.

Problem Q6.3.A **2 POINTS**

memory	contents
M[R8]	7
M[R9]	6

Yes **No**

Problem Q6.3.B **2 POINTS**

memory	Contents
M[R8]	6
M[R9]	7

Yes **No**

Problem Q6.3.C **2 POINTS**

Is it possible for M[R8] to hold 0?

Yes **No**

Now consider the same program, but with two **MEMBAR** instructions.

P1	P2
P1.1: LW R2, 0(R8)	P2.1: LW R4, 0(R9)
P1.2: SW R2, 0(R9)	MEMBAR_{RW}
MEMBAR_{WR}	P2.2: SW R5, 0(R8)
P1.3: LW R3, 0(R8)	P2.3: SW R4, 0(R8)

We want to compare execution of the two programs on our system.

Problem Q6.3.D **4 POINTS**

If both M[R8] and M[R9] contain 6, is it possible for R3 to hold 8?

Without **MEMBAR** instructions? **Yes** **No**

With **MEMBAR** instructions? **Yes** **No**

Problem Q6.3.E **4 POINTS**

If both M[R8] and M[R9] contain 7, is it possible for R3 to hold 6?

Without **MEMBAR** instructions? **Yes** **No**

With **MEMBAR** instructions? **Yes** **No**

Problem Q6.3.F **4 POINTS**

Is it possible for both M[R8] and M[R9] to hold 8?

Without **`MEMBAR`** instructions? **Yes** **No**

With **`MEMBAR`** instructions? **Yes** **No**