Name DAN PORTS

# Computer System Architecture
# 6.823 Quiz #4
# November 22nd, 2006
# Professor Krste Asanovic
# Dr. Joel Emer

Name: DAN PORTS

This is a closed book, closed notes exam.
80 Minutes
13 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 other students who have not yet taken the quiz

| | | |
|---|---|---|
| Writing name on each sheet | 2 | 2 Points |
| Question 1 | 26 | 28 Points |
| Question 2 | 16 | 16 Points |
| Question 3 | 34 | 34 Points |
| **TOTAL** | 78 | **80 Points** |

Name DAN PORTS

# Question 1: Directory-Based Protocol 1 (28 points)

Ben Bitdiddle wants to implement an array increment routine on a 4-processor system. The system uses the directory-based cache coherence invalidate protocol from lecture 18 and handout 11, which is reproduced at the end of the quiz for your reference. Each processor has its own cache and register set.

Assume that the machine is byte-addressable and that the caches start out empty. Also assume that the cache line size is 1 word. In this question we focus on the data caches only.

The array has 128 elements. Assume that in each processor register R1 contains the address of the first element of the array and register R2 contains 32. The following code segments are executed on the four processors. All the code segments are identical except for the offset in the LD and ST instructions.

| Processor A | Processor B | Processor C | Processor D |
|---|---|---|---|
| L1: LD R3, 0(R1) | L1: LD R3, 4(R1) | L1: LD R3, 8(R1) | L1: LD R3, 12(R1) |
| ADDI R3, R3, #1 | ADDI R3, R3, #1 | ADDI R3, R3, #1 | ADDI R3, R3, #1 |
| ST R3, 0(R1) | ST R3, 4(R1) | ST R3, 8(R1) | ST R3, 12(R1) |
| ADDI R1, R1, #16 | ADDI R1, R1, #16 | ADDI R1, R1, #16 | ADDI R1, R1, #16 |
| SUBI R2, R2, #1 | SUBI R2, R2, #1 | SUBI R2, R2, #1 | SUBI R2, R2, #1 |
| BNEZ R2, LOOP | BNEZ R2, LOOP | BNEZ R2, LOOP | BNEZ R2, LOOP |

## *Part A (6 points)*

Assume that Processor A completes the first iteration of the loop before any of the other processors start execution. Provide a sequence of messages that will be generated during the first iteration of the loop on Processor A.

| Instructions | Messages |
|---|---|
| L1: LD R3, 0(R1) | ShReq → home<br>ShRep → $P_A$ |
| ST R3, 0(R1) | InvRep → home<br>ExReq → home<br>ExRep → $P_A$ |

Name DAN PORTS

## *Part B (6 points)* 6

Now assume that all the processors start execution simultaneously. Provide a sequence of messages that will be generated during the first iteration of the loop on Processor A.

| Instructions | Messages |
|---|---|
| L1: LD R3, 0(R1) | Sh Req -> home<br>Sh Rep => PA |
| ST R3, 0(R1) | InvRep -> home<br>Ex Req -> home<br>Ex Rep -> PA ✓ |

## *Part C (6 points)* 6

Louis Reasoner learned from Professor X that longer cache lines improve performance. He decides to change the cache line size to 4 words. When he runs the same code on the four processors, he observes that it takes more time to execute. Explain why this might happen.

With 1-word cache lines, the 4 processors are never accessing data in the same cache line as another processor. With 4-word lines, all are accessing the same cache line, so they will have to request a write-back or invalidate from the other processors on each memory access. ✓

Name DAN PORTS

## *Part D (10 points)* 8

For a cache line size of 4 words, rewrite the code segment for each processor so that the least possible number of messages is generated.

should be 128 256 384

| Processor A | Processor B | Processor C | Processor D |
|---|---|---|---|
| | ADDI R1, R1, R2 | ADDI R1, R1, R2<br>ADD R1, R1, R2 | ADDI R1, R1, R2<br>ADDI R1, R1, R3<br>ADDI R1, R1, R2 |
| L1:<br>LD R3, 0(R1) | | | |
| ADDI R3, R3, #1 | Same | | |
| ST R3, 0(R1) | | | |
| ADDI R1, R1, #4 | | | |
| SUBI R2, R2, #1 | | | |
| BNEZ R2, L1 | | | |

Name DAN PORTS

# Question 2: Directory-Based Protocol 2 (16 Points)

Consider a multiprocessor system with a directory-based cache coherence invalidate protocol which preserves sequential consistency. Each processor has its own cache and register set, and the cache lines are 4 words long.

## *Part A (8 Points)* 9

A cache PP might receive an ExRep message when it is in the C-Nothing state. It dequeues the message, changes the state of the cache line to C-Exclusive and updates the cache data with the data in the message. Describe a scenario that will cause this situation.

Suppose A pretetches a cache line, then decides to store to it, generating an Ex Req. The pretetch generates a Sh Rep, which will put A into the C-Shared state. Then the home directory processes processor B's Ex Req for the line before A's. This generates a Inv Req to A, which will put it in the C_nothing state. When A's ExReq is processed, the home will (after forcing a writeback from B) send a ExRep to A. ✓

Name DAN PORTS

## *Part B (8 Points)* 8

We have decided to slightly modify our directory-based cache coherence invalidate protocol in order to speed up stores. On a store miss a cache PP changes its state to C-Exclusive instead of C-Pending and sends an ExReq message to home. Upon receiving an ExReq message, home either sends out InvReq messages to all the cache PPs which have shared copies of the cache line, or an InvReq message to the cache PP which has an exclusive copy of the cache line. Upon receiving the InvRep message(s), home moves into the W(id) state. If no processor has either a shared or an exclusive copy of the cache line, it moves into the W(id) state directly. It does not send an ExRep message to the cache PP which requested the exclusive copy of the cache line.

Do you think the directory-based protocol will function correctly with this modification? Explain your reasoning.

No. Suppose processor A holds a certain cache line exclusively and has modified all words in it. If processor B attempts to store one word in the line, it will cause processor A to invalidate its cache, but will never receive the modifications to the other words in the line (which would have been sent in the ExRep).. ✓

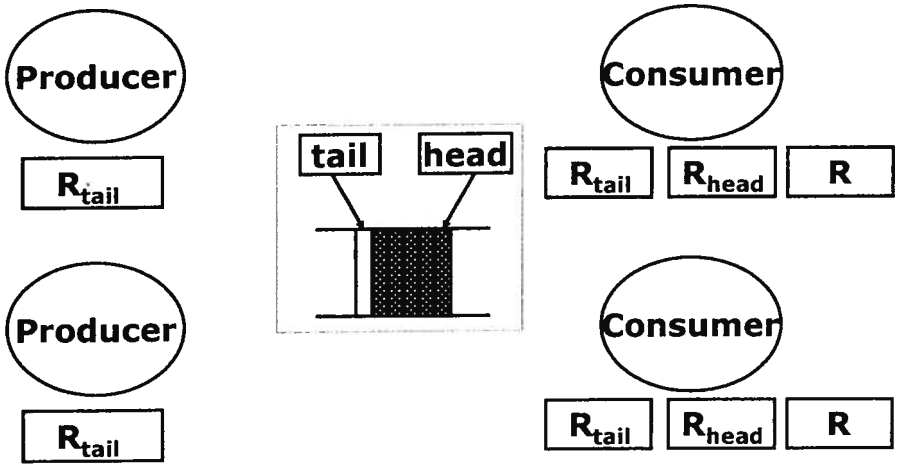
Name DAN PORTS

# Question 3: Load-reserve & Store-conditional (34 points)

In this question, we are going to experiment with two synchronization instructions: Load-reserve and Store-conditional. The following are the definitions of the instructions given in Lecture 16.

| **Load-reserve R, (m):** | **Store-conditional (m), R:** |
|---|---|
| <flag, addr> ← <1, m>;<br>R ← M[m]; | if <flag, addr> == <1, m><br>then cancel other procs' reservation on m;<br>M[m] ← R;<br>status ← succeed;<br>else status ← fail; |

In this question, we will write code for multiple producers and multiple consumers sharing a single queue.



Name DAN POETS

## *Part A (9 points)*

```
try:    Load-reserve R_head, (head)
spin:   Load R_tail, (tail)
        if R_head == R_tail goto spin
        Load R, (R_head)
        R_head = R_head + 1
        Store-conditional (head), R_head
        if (status==fail) goto try
        process(R)
```

The above code is for the consumers. It is the same as the one given in Lecture 16. Assume that multiple consumers execute the code at the same time and there are many items to be consumed. Can deadlock, livelock or starvation occur? Explain your reasoning for each case.

Deadlock cannot occur because there are no instructions that block, so it's always possible to keep executing (even if it's just spinning).

Livelock cannot occur. One processor will always succeed in executing the store-conditional (since the reservation is only cleared by another processor's successful store). Hence, one processor will make progress as long as there are items in the queue.

Starvation can occur. If one processor consumes an item, processes it, and re-executes the load-reserve before any other one can, it will consume the next item too. There's no mechanism for ensuring fairness.



9

Name Dan Ports

## *Part B (5 points)*

Ben Bitdiddle is responsible for writing the code for the producers. He looks at the code in Part A and then modifies the code for the producer in Lecture 16 to give the following.

```
try:    Load-reserve R_tail, (tail)
        Store (R_tail), x
        R_tail = R_tail + 1
        Store-conditional (tail), R_tail
        if (status==fail) goto try
```

Alyssa P. Hacker looks at Ben's code and says the code won't work. What's the problem?

Consider the following sequence of events:

A: LD-reserve
ST -
Rtail = Rtail + 1
ST - conditional (succeeds)

B: LD-reserve
ST
Rtail = Rtail + 1
ST - conditional (fails)

Processor A's execution succeeded, but the value in the queue is processor B's. ✓

## *Part C (12 points)*

Rewrite the code in Part B to make it work correctly. You are allowed to introduce additional shared variables.

```
spin: LD-reserve R_flag (flag)
      if (R_flag == 1) goto spin
      store-conditional (flag), 1
      if (status == fail) goto spin

critical section {
      LD   R_tail, (tail)
      R_tail = R_tail + 1
      ST   (R_tail), x
      ST   (tail), R_tail
      ST   flag, 0
}
```



12

Name DAN PORTE

## *Part D (8 points)*

Ben now extends his in-order 5-stage pipeline machine to support load-reserve and store-conditional. Prior to this, Ben has already modified his original direct-mapped cache design into a snoopy cache with invalidate protocol. He remembers from the 6.823 lecture that the implementation of Load-reserve and Store-conditional matches very well with the invalidate protocol. To minimize hardware cost, he uses his existing cache design to track the reservation instead of adding additional registers. In his scheme, when the processor executes a load-reserve instruction, it will bring in an exclusive copy of the address to its cache. Later on, when the processor executes a store-conditional on the same address, it will succeed only if the exclusive copy is still in the cache (i.e., if the store hits in the cache).

After the modification, Ben runs the producer/consumer programs from part A and part C on his multiprocessor system. He discovers that no consumer is able to consume any item. Suggest two possible causes for this to happen.

1) During the instructions between the LD_reserve and ST_conditional, the exclusive copy is evicted from the cache, causing the ST_conditional to always fail

2) The following sequence occurs:

| A | B | |
|---|---|---|
| LD Res | | Each time a processor tries to execute ST cond, the other processor invalidates its cache by reexecuting a LD res |
| | LD Res | |
| ST cond | | |
| LD res | | |
| | ST cond | |
| | LD res | |
| ST cond | | |
| LD res | | |
| | ST cond | |
| | LD res | |
| ⋮ | ⋮ | |

✓



8