15-418/618 Spring 2021
Exercise 5

Assigned:	Wed., Mar. 17
Due:	Fri., Mar. 26, 11:00 pm

Overview

This exercise is designed to help you better understand the lecture material and be prepared for the style of questions you will get on the exams. The questions are designed to have simple answers. Any explanation you provide can be brief—at most 3 sentences. You should work on this on your own, since that's how things will be when you take an exam.

You will submit an electronic version of this assignment to Canvas as a PDF file. For those of you familiar with the LaTeX text formatter, you can download the template and configuation files at:

http://www.cs.cmu.edu/~418/exercises/ex5.tex
http://www.cs.cmu.edu/~418/exercises/config-ex5.tex

Instructions for how to use this template are included as comments in the file. Otherwise, you can use this PDF document as your starting point. You can either: 1) electronically modify the PDF, or 2) print it out, write your answers by hand, and scan it. In any case, we expect your solution to follow the formatting of this document.

Problem 1: Memory Consistency

Assume the following program segments are executed on three processors of a multiprocessor machine. Initially before execution, all variables are equal to 0.

P1	P2	P3
`E1a: A = 1`	`E2a: u = A`	`E3a: v = B`
	`E2b: B = 1`	`E3b: w = A`

A. There are totally 8 possible final states of `u`, `v`, `w` as listed. For each one, indicate whether it is valid or invalid under a sequential consistency model.

Case	Final states	Valid? (Y/N)
Case 1:	u = 0 v = 0 w = 0	Yes
Case 2:	u = 0 v = 0 w = 1	Yes
Case 3:	u = 0 v = 1 w = 0	Yes
Case 4:	u = 0 v = 1 w = 1	Yes
Case 5:	u = 1 v = 0 w = 0	Yes
Case 6:	u = 1 v = 0 w = 1	Yes
Case 7:	u = 1 v = 1 w = 0	No
Case 8:	u = 1 v = 1 w = 1	Yes

B. Choose one of the final states that you think is invalid under sequential consistency. Prove that it is invalid. (Refer to the operations by the event labels `E1a`, `E2b`, etc.)

Case 7 is invalid. Consider the following execution:

- P1 executes and A is set to 1.
- P2 executes and u is set to 1(valid). P3 executes and v is set to 0 (unvalid).
- P3 executes and v is set to 1, w is set to 1.

C. To maintain the same possible final states (as in sequential consistency) under weaker memory consistency model, one way is to add fences that guarantee that all reads/writes prior to the fence complete before any read/write after the fence. There are 8 possible placements to add fences:

P1	P2	P3
$Fence_1$	$Fence_2$	$Fence_3$
`E1a: A = 1`	`E2a: u = A`	`E3a: v = B`
$Fence_4$	$Fence_5$	$Fence_6$
	`E2b: B = 1`	`E3b: w = A`
	$Fence_7$	$Fence_8$

To minimize the number of necessary fences (while maintaining sequential consistency), where would you place the fences? List a minimal set of fences, and argue that removing any one of these could lead to a consistency violation.

Fence8 should be enough to maintain sequential consistency. Only Case7 should not exist. If we fence after E3b, then E3a cannot execute without E3b. So consistency is maintained.

D. Suppose you could use also use a storage fence or a load fence. Could you replace any of the full memory fences in your list with one of these?

I have one fence in my list. I can replace it with a storage fence for w, and E3b, E3a will be executed in order.

Problem 2: Cache Coherency

You are hired by Intel to design an invalidation based cache-coherent processor with a large number of cores. The main application for this processor will be to train a chess AI by playing games against itself for experiments. It is critical that it doesn't play too many games during training to maintain the validity of the experiment, so we store how many games are played in a shared counter:

```
int num_games = 100000;
int games_played = 0;

// This code is run on each core
while (true) {
  // Atomically get value of count prior to increment, and
  // write incremented value to games_played
  int val = atomic_add(&games_played, 1);

  if (val > num_games) break;
  play_game()
}
```

Unsure on how to design the cache coherence of this processor, you check with one of the senior designers. She suggests that you should use a bus-based, snooping coherence implementation, rather than a directory-based protocol, because the broadcasting it performs would be more efficient in this case. Do you agree or disagree? Why or why not?

I agree with senior designer because all cores need to know the value of games played. So a bus-based protocol is more efficient because of this equal workload distribution.