CSC 458: Assignment #3

Due on Tue, Feb 25, 2014

Kai Shen 15:25am

Qiyuan Qiu

Contents

Problem 1

Write-After-Read case:

```
add r2, r1, 4
load r1, memaddr
```

Here the r1 location is read in the first instruction and later written in the second instruction.
If we map the first r1 to R3, and the second r1 to R4, the code will be turned into

```
add r2, R3, 4
load R4, memaddr
```

Now these two instructions can be issued simultaneously because they are now independent of each other.
Write-After-Write case:

```
write r2, memaddr1
write r2, memaddr2
```

Here both instructions write to memory location r2. If we map the first r2 to R3 and the second r2 to R4, we get the following code.

```
write R3, memaddr1
write R4, memaddr2
```

Now these two instructions are independent to each other.

Problem 2

Assume that when we talk about cache coherence protocol, the protocol is used for a SMP. SMP means that there are multiple processors; each processor has its own private cache; all processors share a main memory by sharing the same bus.

Update here means that the cache policy is write through. When a processor writes a value to its cache, the write is propagated to main memory so that any other processor is aware of this change.

Invalidation here means that the cache policy is write back. When a processor writes a value to its cache, this change is not right away written to the shared main memory. Rather the change is written the shared main memory when the cache is replaced.

The **advantage** for updating is that it is easier to implement.

The **disadvantage** of updating is that because every cache write goes to bus, the bandwidth requirement for bus will be high, which limits the number of overall processors working on the same chip.

Problem 3

No. In the following scenario, both processors will run in critical section. Assume the caches are write back caches.

Initially, both P1 and P2 have flag1 = 0 and flag2 = 0 in their own private cache.

Now lets focus on P1. P1 write flag1 = 1, turn = 2. But because of write back cache property, both of these write do not propagate to main memory. In P2's private cache , flag1 has the stale value 0.

Similarly, the write P2 performs does not update the value of flag2 in P1's private cache either.

As the result, when the while in P1 checks the value of flag2 it sees that flag2 in its cache is still 0 so it goes into the critical section. When the while in P2 checks the value of flag1 it has a hit in its cache and believes that flag1 is 0 and goes into the critical section.

Hence, the above program does not guarantee mutually exclusive executions of the critical section.

Problem 4

Sequential memory consistency(SC) subsumes coherence. Coherence model requires the write order to same memory location shows up the same to all processors. SC is an stronger assumption in a sense that it requires all the write, even to different memory locations, appears to all other processors in a consensus order. SC also ensures the individual program order for each processor.

As for this particular problem. There is only one possible output for P3, which is 1.(Or do not execute print(A))

The reason being that, due to sequential memory consistency, B is set to 1 if and only if the value of A has been set to 1. Hence P3 either does not output anything or it printout 1.

Problem 5

Problem 5: (a)

Because the test_and_set is a write and due to the property of sequential memory consistency, all test_and_set from all processors are ordered. In that order the next test_and_set cannot be issued until the current one finished. Therefore ensures mutual exclusive access to critical section.

Problem 5: (b)

He is right.
If some processor runs within the first while means that it is the only one owns the critical section. The test_and_set instruction already takes care of setting the value of location to locked.

Problem 5: (c)

He is right.
Assuming a shared memory model. Each processor has its own private cache. The test_and_set is a read-modify-write instruction. That means every time it is used, the memory coherence protocol have to notify all other processors to make sure the order of all write operation appears all the same to them.
The second while loop is just a read. This can easily have a hit in the processors private cache, which lead to much better performance.

Problem 6

```
def test_and_set:
    if( compare_and_swap(location,locked, locked) == 1):
        return locked
    else:
        return unlocked
```