CS152 Section 12

Q1: Memory Consistency Models

Consider the following two threads executing on two different cores. Assume that memory locations A, B, and C are all initialized to zero.

	P1:		**P2:**
	`li x1, 1`		`li x1, 2`
I1	`lw x2, A`	**J1**	`sw x1, B`
I2	`sw x1, C`	**J2**	`lw x2, C`
I3	`lw x3, B`	**J3**	`sw x1, A`

We are interested in the final values of P1.x2, P1.x3, and P2.x2.

Q1.1: Sequential Consistency
Give all possible sets of values of P1.x2, P1.x3, and P2.x2 under sequential consistency (SC).

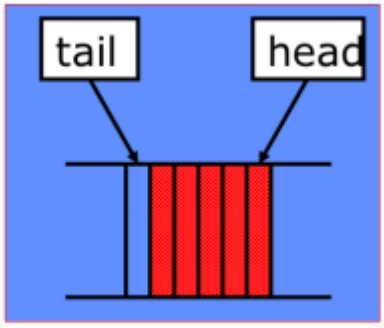
Q1.2: W→R Relaxation
Give all new possible sets of values if we relax Write → Read ordering constraints and the instruction orderings that caused them.

Q1.3: W→W Relaxation
Give all new possible sets of values if we relax Write → Write ordering constraints and the instruction orderings that caused them.

Q1.4: R→R and R→W Relaxation
Give all new possible sets of values if we relax Read → Read and Read → Write ordering constraints and the instruction orderings that caused them.

Q2: Synchronization

We want to write a multithreaded program that uses a producer-consumer model. A producer thread computes some value and sends it to a consumer thread through a queue. The queue is an array in memory with a head pointer and a tail pointer. The producer pushes an item onto the queue by writing to the address pointed to by the tail pointer and then incrementing the tail. The consumer pulls an item from the queue by reading from the address pointed to by the head pointer and then incrementing the head. The queue is empty if the head pointer and tail pointer are the same.



Assuming that the queue is infinitely long, and each item is eight bytes, the assembly code for the producer and consumer program is as follows.

Producer	Consumer

`# x1: address of tail pointer`
`# x2: data to be written`
 	
`  ld   x3, 0(x1)`
`  sd   x2, 0(x3)`
`  addi x3, x3, 8`
`  sd   x3, 0(x1)`
	`# x1: address of tail pointer`
	`# x2: address of head pointer`
	
	`  ld   x3, 0(x2)`
	`spin:`
	`  ld   x4, 0(x1)`
	`  beq  x3, x4, spin`
	`  ld   x5, 0(x3)`
	`  addi x3, x3, 8`
	`  sd   x3, 0(x2)`
	`# then process x5`

This code will be correct if the memory system is sequentially consistent and there is only one producer and one consumer.

Q2.1: Relaxed Memory Model

Would this still be correct if we had a relaxed memory model? What problems could occur?

Q2.2: Fences

What is the minimum set of fence instructions that needs to be added to make these programs work under a relaxed memory model?

Q2.3: Atomic Synchronization Primitives

Now assume we have multiple producers sharing the same queue. We want to rewrite the producer code to make it thread safe. Assume we no longer have to worry about the order of storing to the queue vs. storing the tail pointer.

We can use an atomic fetch-and-add, a compare-and-swap, or a test-and-set instruction as our synchronization primitive. Write the producer code using each of these. Assume as before that register x1 contains the address of the tail pointer and that register x2 contains the data to be stored. Pseudocode for these primitives is shown below.

Atomic Add	Compare and Swap	Test and Set
amoadd rd, rs1, (rs2)	**cas rd, rs1, rs2, (rs3)**	**ts rd, (rs)**

rd <= M[rs2]
M[rs2] <= rd + rs1
	if (rs1 == M[rs3])
	  M[rs3] <= rs2
	  rd <= 1
	else
	  rd <= 0
		rd <= M[rs]
		M[rs] <= 1

Atomic Add	

Compare and Swap	
Test and Set	

Q2.4: Efficiency

Which of these methods is most efficient (least number of memory transactions)? Which is the least efficient?