**Memory Models Worksheet 1:**

**Hardware and Relaxed Consistency**

**“how to frighten small children”**

1. Please fill out and label the dependencies between these instructions.

(dependency types include: [m]emory, [p]rogram order, [d]ata (or register))

1. load [a] -> r1

2. 20 / r1 -> r1

3. store r1 -> [c]

4. load [c] -> r2

5. r1 + r2 -> r2

6. store r2 -> [c]

1. In the above code snippet, what is the single longest chain of dependent instructions that cannot be broken by out-of-order execution? (e.g. 1 -> 2 -> 4)
2. Imagine you have been given the spinlock definition below. On which memory models we’ve discussed in class does the below code enforce mutual exclusion? Give an example of how the blow code breaks on that memory model, or explain how the below code will not violate mutual exclusion on any of the memory models we’ve discussed. (hint: try writing out code that uses the lock)

NOTE:

xchg (x, y) is

implemented as:

 atomic {

 old = x;

 x = y;

 return old

 }

and enforces no additional ordering constraints.

**Lock()** {

 do {

 tmp = xchg(unlocked, 0);

 if (tmp != 1) {

 while (unlocked != 1) ;

 }

 } while (tmp != 1)

}

**Unlock()** {

 unlocked = 1

}

1. Assume you have the 3 fence instructions described on the last sheet of this handout, and the TSO architecture discussed in class. What values can be printed by the following code snippets, and why:

**Initial**: a = 0, b = 0

* 1.

**Thread 2**

b = 1

SFENCE

print a

**Thread 1**

a = 1

SFENCE

print b

* 1.

**Initial**: a = 0, b = 0

**Thread 2**

b = 1

LFENCE

print a

**Thread 1**

a = 1

LFENCE

print b

**Initial**: a = 0, b = 0

**Thread 1**

a = 1

MFENCE

print b

**Thread 2**

b = 1

MFENCE

print a

1. Your friend Billy-Jean comes to you and (very aware that you are the one who can answer her memory model concerns) tells you that she’s come up with a wonderful new architecture that she believes will support SC. She’s going to construct an out-of-order processor with a combined FIFO load/store queue, but she’s adding one optimization. She is going to speculatively execute all the instructions in the out-of-order processor dependent on waiting loads as if the load had returned 0. If when the load finally gets to the head of the queue and gets its value from memory that value isn’t 0, the processor will discard any dependent execution within the processor and re-run that execution with the real returned load.

Assuming performance, and single-threaded correctness are not a concern (she’s implemented her complex speculation idea correctly), will her described processor implement SC? Why or why not? (Hint, what is the effect of speculation on instruction ordering)

**X86 Fence Instructions (simplified from x86 instruction descriptions)!**

**LFENCE**

Performs a serializing operation on all load-from-memory instructions that were issued prior the LFENCE instruction. Specifically, LFENCE does not execute until all prior instructions have completed locally, and no later instruction begins execution until LFENCE completes. In particular, an instruction that loads from memory and that precedes an LFENCE receives data from memory prior to completion of the LFENCE.

**SFENCE**

Orders processor execution relative to all memory stores prior to the SFENCE instruction. The processor ensures that every store prior to SFENCE is globally visible before any store after SFENCE becomes globally visible. The SFENCE instruction is ordered with respect to memory stores, other SFENCE instructions, and MFENCE instructions. It is not ordered with respect to memory loads or the LFENCE instruction.

**MFENCE**

Performs a serializing operation on all load-from-memory and store-to-memory instructions that were issued prior the MFENCE instruction. This serializing operation guarantees that every load and store instruction that precedes the MFENCE instruction in program order becomes globally visible before any load or store instruction that follows the MFENCE instruction. The MFENCE instruction is ordered with respect to all load and store instructions, other MFENCE instructions, any LFENCE and SFENCE instructions. MFENCE does not serialize the instruction stream.