Exercises - The x86-TSO model

1. Peterson algorithm is a classic solution to the *mutual exclusion* problem: in all executions, the instructions of the critical sections of the two threads are not interleaved.

```
flag0 = false;
flag1 = false;

flag0 = true;                        flag1 = true;
turn = 1;                            turn = 0;
while (flag1 && turn == 1);     ||     while (flag0  && turn == 0);
// critical section                   // critical section
   ...                                  ...
// end of critical section            // end of critical section
flag0 = false;                       flag1 = false;
```

   (a) Assume a sequentially consistent execution model and explain informally why the two threads cannot be inside the critical section at the same time.

   (b) Does Peterson algorithm guarantee mutual exclusion if executed on a multiprocessor machine where store buffers are observable (e.g. x86)? In case, where would you put memory barriers to ensure the correctness of the algorithm?

2. In terms of the formal SC semantics:

   (a) Give two different transitions, with derivations using the rules in the notes, of the process `t1:(x=(y=z))`.

   (b) Give a complete transition sequence of the whole-system state `<t1:(x=(y=z)), {x=1,y=2,z=3}>`. Is it unique?

   (c) By enumerating the possible whole-system transitions (without giving their derivations in detail), or otherwise, prove that `<t1:(x=1);y | t2:(y=1);x, {x=0,y=0}>` cannot reach a state of the form `<t1:0 | t2:0, M>`.

3. Consider this x86 example. Initially all registers and `[x]` and `[y]` are `0`.

```
Thread 0         Thread 1          Thread 2
MOV [x] <- 1     MOV EAX <- [x]    MOV [y] <- 1
                 MOV EBX <- [y]    MOV ECX <- [x]
```

   Finally: `Thread 1: EAX=1`, `Thread 1: EBX=0`, `Thread 2: ECX=0`.

   (a) Is this allowed with respect to an SC semantics?

   (b) Prove whether or not it is allowed with respect to the x86-TSO abstract machine.

4. Download and install the `litmus` tool from `http://diy.inria.fr/sources/litmus-5.01.tar.gz` (you need OCaml > 3.12.0; documentation available from `http://diy.inria.fr/doc/litmus.html`). Test the processor of your laptop against the following examples:

```
X86 SB                                  X86 MB
{ x=0; y=0; }                           { x=0; y=0; }
 P0          | P1          ;             P0          | P1          ;
 MOV [x],$1  | MOV [y],$1  ;             MOV [x],$1  | MOV EAX,[y] ;
 MOV EAX,[y] | MOV EAX,[x] ;             MOV [y],$1  | MOV EBX,[x] ;
exists (0:EAX=0 /\ 1:EAX=0)             exists (1:EAX=1 /\ 1:EBX=0)
```