Parallel Computer Systems
Lab session 3

Caroline De Brouwer & Titouan Vervack
Group 12

Question 1

The counter does not return to its initial value because there are data race conditions.

Question 2

Yes, the counter returns to its initial value because the threads never execute critical sections at the same time.

Question 3

a) Volatile will always fetch the variable from main memory even if it doesn't appear to have changed. This is used because the variable may have been edited by another thread.
b) The critical section can't be executed by both threads at once, therefore they need to take turns.
c) No, there is inconsistency: out-of-order operations can make the threads execute critical sections at the same time. For example:
- flag[i] is false
- Thread j stores flag[j]=false
- Thread i loads flag[j], doesn't go into the while loop
- Thread j loads flag[i], doesn't go into the while loop
- Thread i stores flag[i]=true

Question 4

The first MFENCE is before the while loop. If it's not there the example above can occur, in which the flag gets read by the other thread before it has been stored. The second one is at the end of the if-block, for the same reason.

Question 5

There is inconsistency when using the non-atomic instructions, because there are again data race conditions. This does not happen with atomic instructions because when an instruction is executed atomically another thread can not do anything in between the reading, incrementing (or decrementing) and writing of the variable.

Question 6

The same problem occurs with the non-atomic instructions: memory instructions are reordered and the program will not execute correctly. With the atomic compare-and-exchange instructions, the incrementing or decrementing does not execute when the data it tries to modify has changed, so it will only execute when it can modify the value correctly. Because of this there is no more inconsistency.

Question 7

With the compare and exchange function you can do whatever operation you want instead of just incrementing and decrementing, for example you could do an atomic multiplication.

Question 8

Using critical sections with pthreads takes 0.2259 sec on average, atomic_incdec takes 0.0487 sec on average. The pthreads with critical sections are much slower because they have the overhead of managing the flag and turn variables, and most of the time one thread is actively waiting on the other.

Question 9

The algorithm with atomic operations averagely takes 0.0487 sec, the one with non-atomic operations takes 0.0030 sec. The reason is that the non-atomic operations don't have to wait for the other thread to finish before they can start incrementing or decrementing.