Design of Parallel and High Performance Computing
HS 2017
Torsten Hoefler, Markus Püschel **Homework 5**
Department of Computer Science
ETH Zurich

A new lock?

Does the following code ensure that at any time, at most one thread is in the critical region if we assume sequential consistency? Use sequential consistency to prove your answer.

The variables me and $other$ are thread-local, the variables $want[0]$, $want[1]$ and $turn$ are shared. The variables $want$ and $turn$ are initially zero, me contains the thread id($\in 0, 1$) for each thread, while $other$ contains the value $1 - me$.

```
want[me] = 1;
while (turn != me) {
    while (want[other]) {}
    turn = me;
}
// CR
want[me] = 0;
```

Peterson Lock on x86

Prove that the following implementation of the Peterson lock is correct in the x86 memory model. The variables me and $other$ are thread-local, the variables $want[0]$, $want[1]$ and $victim$ are shared. The variables $want$ and $victim$ are initially zero, me contains the thread id($\in 0, 1$) for each thread, while $other$ contains the value $1 - me$.

Use the following properties of the x86 memory model (cf. Sec. 8.2.2. of the Intel Intel 64 and IA-32 Architectures Software Developers Manual, Vol 3A):

- Writes to memory are not reordered with other writes.
- Reads are not reordered with other reads.
- Reads cannot pass earlier LFENCE and MFENCE instructions.
- Writes cannot pass earlier LFENCE, SFENCE, and MFENCE instructions.
- MFENCE instructions cannot pass earlier reads or writes.
- Stores are not reordered with older loads.

```
want[me] = 1;
victim = me;
asm("mfence");
while (want[other] && (victim == me)) {};
// CR
asm("mfence");
want[me] = 0;
```

Can you explain the reason for the mfence instruction in the unlock phase?