York University | EECS 4101/5101 | August 4, 2020

Homework Assignment #9
Due: August 12, 2020 at 12:00 noon

1. In this question, we consider the problem of building a shared implementation of a stack. We will only be concerned with stacks that are limited to contain at most N items at any time. In other words, you can assume (as a precondition of Push) that no process will try to Push a value on to a stack that already contains N elements.

(a) Consider the following implementation from read/write registers. It uses a shared array $A[1..N]$ of read/write registers and a shared read/write register Top. Assume the stack is initially empty and that Top is initialized to 0. Each process also uses a *local* variable t.

```
1   Push(x)
2       t ← Top.read
3       Top.write(t + 1)
4       A[t + 1].write(x)
5       return Ack
6   end Push

7   Pop
8       t ← Top.read
9       if t = 0 then return Empty
10      else
11          Top.write(t − 1)
12          return A[t].read
13      end if
14  end Pop
```

Is this implementation correct if only one process accesses the stack at a time? Briefly justify your answer.

(b) Now, suppose that the implementation in part (a) can be accessed by two processes concurrently. Consider an execution where each of the two processes perform a Push operation followed by a Pop. Furthermore, suppose both Push operations execute line 2 before either operation executes line 3. Construct the rest of the execution so that the execution is *not* linearizable. Conclude that the implementation in part (a) is not linearizable.

(c) Now suppose we discover that the hardware that we are using for our stack implementation also provides two machine instructions called fetch&inc and fetch&dec. A fetch&inc on a memory location allows us to atomically fetch the old value stored in the memory location and increment it by 1. In other words, a fetch&inc instruction changes the state of the memory location from the value v to $v+1$ and returns v. Similarly, a fetch&dec instruction atomically fetches the old value stored in the memory location and decrements it by 1. (Assume that the value in the memory location is a non-negative integer, and that a fetch&dec on a memory location whose value is 0 returns 0 and does not change the value stored there, so that the value never becomes negative.) We use these new instructions to rewrite our implementation in part (a) as follows.

```
15  Push(x)
16      t ← Top.fetch&inc
17      A[t + 1].write(x)
18      return Ack
19  end Push
```

```
Pop
    t ← Top.fetch&dec
    if t = 0 then return Empty
    else return A[t].read
    end if
end Pop
```

Is this a linearizable implementation of a stack (for two processes)? Show that your answer is correct.