This handout describes the implementation of a shared-memory queue that supports a single producer thread and multiple consumer threads. For simplicity, we assume the queue has infinite space. The queue uses the atomic compare-and-swap (CAS) instruction, defined as follows:

\[
\text{CAS old, new, Imm(base)} \quad \text{atomically loads the value at the effective memory address and compares it with the value stored in register old. If both values are equal, it updates the memory location with the value stored in register new. If both values are not equal, it updates the value in old with the value loaded from memory.}
\]

The queue stores single-word messages. The code for producer and consumers are shown below, with memory operations highlighted in bold.

**Code for producer to enqueue a message:**

```assembly
# R1 - contains message to enqueue
# R2 - contains address of the tail pointer of the queue

P1: LD R3, 0(R2)     # get tail pointer
P2: ST R1, 0(R3)     # write message to tail
P3: ADD R3, R3, 4    # update tail pointer
P4: ST R3, 0(R2)     # update tail pointer
```

**Code for consumer to dequeue a message:**

```assembly
# R1 - contains dequeued message after code finishes
# R2 - contains address of the head pointer of the queue
# R3 - contains address of the tail pointer of the queue
# R4 - contains address of the head pointer write lock
# R5 - contains value 1

C1: SpinLock: MOV R6, R0     # set R6 to 0
C2: CAS R6, R5, 0(R4)        # try to acquire lock
C3: BNEZ R6, SpinLock
C4: LD R7, 0(R2)             # get head pointer
C5: Retry: LD R8, 0(R3)      # get tail pointer
C6: BEQ R7, R8, Retry        # is there a message?
C7: LD R1, 0(R7)             # read message from queue
C8: ADD R7, R7, 4            # update head pointer
C9: ST R7, 0(R2)
C10: ST R0, 0(R4)           # release lock
```