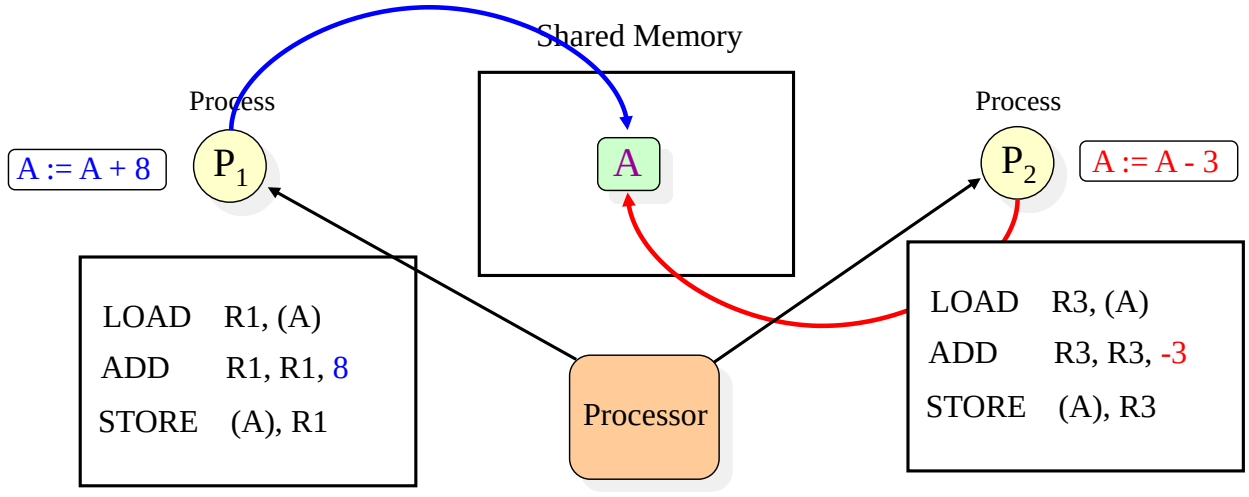
CS314 Operating System
Spring 2024
Exercise Questions on February 6th (PART I)

EXERCISE #1

Suppose that a processor (single-core processor) is executing the following two instructions in the two processes. If the initial value of variable 'A' is 100 at the beginning (before this processor executes any of the two statements), what are the possible values in 'A' after this processor executes the two statements (once for each)? Show all the possible values in 'A'.



EXERCISE #2

An implementation of a circular FIFO queue for multiple producers and consumers are attached in the appendix (it is the one we designed in our lecture). Regarding the implementation:

Question #1: Is it necessary to have "wait (mutex)" and "signal(mutex)" in the producer processes? If NO, explain why not. If YES, explain why we need them.

Question #2: Is there any merit in using two different "mutex semaphore" (one for the producers and the other for the consumers)? If yes, describe what is the merit? If not, why not?

Question #3: Is the implementation "starvation free" for both producers and consumers? If NO, explain why it is not. If YES, explain why we it is.

APPENDIX:

```
#define   N   100                    // the queue size

shared memory int CFQ [N];     // the circular FIFO queue
shared memory int TOP = 0;     // pointer to the top of the queue
shared memory int TAIL = 0;    // pointer to the tail of the queue

semaphore mutex = 1;           // a mutex semaphore
semaphore empty = N;           // a counting semaphore
semaphore full = 0;            // a counting semaphore
```

```
void producer (void)
{
     int new_item;              // place holder for a new item to insert

     while (TRUE)
     {
        new_item = produce_new_item( ); // generate a new item

        wait (empty);           // make sure the queue is NOT full
        wait (mutex);           // I should be the only one

        insert(CFQ, TAIL, new_item);  // insert the new item to the queue
        TAIL = (TAIL + 1) % N;      // update the Top pointer

        signal(mutex);          // I am done!
        signal(full);           // (full)  = (full) + 1
     }
}
```

```
void consumer (void)
{
     int new_item;              // place holder for a new item to insert

     while (TRUE)
     {
        wait (fill);            // make sure the queue has at least one slot
        wait (mutex);           // I should be the only one

        new_item = remove(CFQ, TOP);  // remove one item from the queue
        TOP = (TOP + 1) % N;        // update the Top pointer

        signal(mutex);          // I am done!

        signal(empty);          // (empty)  = (empty) + 1

        use_the_new_item(new_item); // use the new item
     }
}
```