2. Locking

c)

i. For flag.s, the final count is only 26. This is because flag.s doesn't apply a mutex to the %ax variable or to the flag. Because of this, multiple threads can end up accessing and manipulating either value at the same time, which leads to multiple threads executing the critical section at once, which results in incorrect incrementation, which results in 26 instead of 30 because of the lack of locks.

Ii. For test-and-set.s, the value ends up at 30. The condition for moving into the critical section is that %ax must equal 0, and the only way that %ax can equal 0 is if mutex equals 0 because the values of %ax and mutex get swapped (%ax gets put into memory at the address of mutex, and mutex's value gets loaded into %ax). Because of this memory exchange, only one thread can have access to the value at mutex at a time. So, the critical section is only accessible by one thread at a time. After the critical section is executed, 0 gets put into the memory address of mutex, thus releasing the lock and allowing another thread to attempt to execute the critical section.

yield.s functions very similarly to test-and-set.s, with the only exception being that if a thread finds that %ax == 0, it will switch to the next thread in the runqueue.

d)

Total Instructions Executed for each Program:

flag.s	test-and-set.s	yield.s
405	596	561

flag.s has the lowest number of instructions because sometimes different threads end up accessing the flag variable at the same time and there are no checks in place to ensure that this doesn't happen. Those checks end up being a bit more costly because they add to the total number of instructions. test-and-set and yield do have checks in place, so whenever a thread finds that %ax doesn't equal 0, it has to go back to .acquire and execute that section again and again until it finds that %ax does equal zero, thus adding to the total number of instructions. I think that yield has fewer than test-and-set because whenever a thread yields, it's switching states from running to ready, and while it's ready it's not executing at all. In test-and-set, the thread is continuously checking whether %ax equals 0, whereas when a thread yields to another thread, the thread that yielded doesn't execute again until it comes back up in the runqueue.

(continued on next page)

e)

Number of instructions with varying interrupt frequencies:

	flag.s	test-and-set.s	yield.s
i = 2	333	717	675
i = 5	405	597	561
i =10	438	485	435
i = 15	579	657	495

Count values of each program with varying interrupt frequencies:

	flag.s	test-and-set.s	yield.s
i = 2	10	30	30
i = 5	26	30	30
i =10	29	30	30
i = 15	30	30	30

Interestingly, for each program the number of instructions goes down until a certain point when it starts to go back up again.

As for the count values, test-and-set and yield both gave the same values regardless of interrupt frequency. This makes sense because of the locking in place in each program. flag, however, gets closer to the goal as the number of interrupts increases.