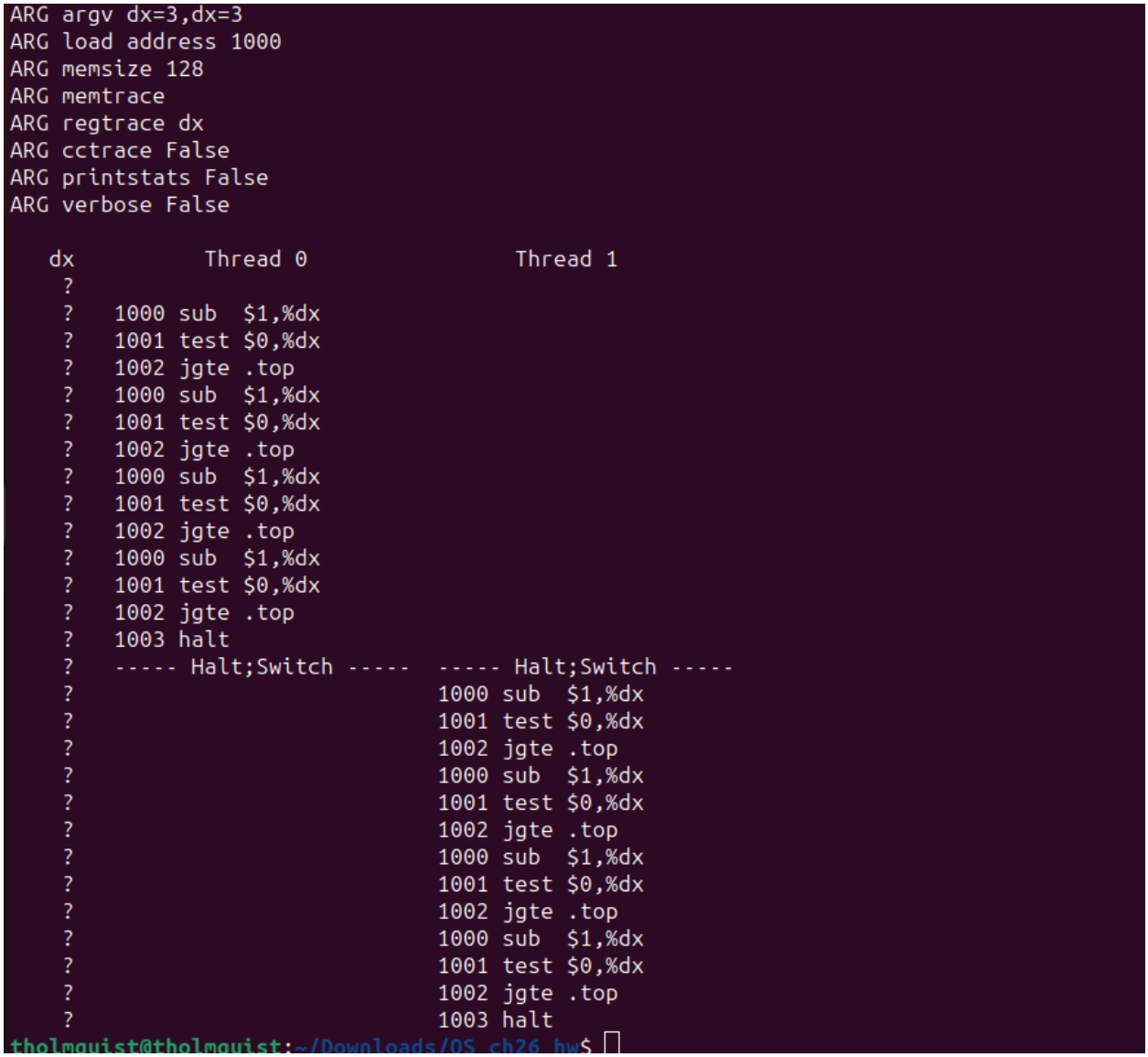
Ch.26 Simulation hw
Tyler Holmquist

1.

```
tholmquist@tholmquist:~/Downloads/OS_ch26_hw$ ./x86.py -t 1 -p loop.s -i 100 -R dx
ARG seed 0
ARG numthreads 1
ARG program loop.s
ARG interrupt frequency 100
ARG interrupt randomness False
ARG argv
ARG load address 1000
ARG memsize 128
ARG memtrace
ARG regtrace dx
ARG cctrace False
ARG printstats False
ARG verbose False

   dx          Thread 0
    ?
    ?   1000 sub  $1,%dx
    ?   1001 test $0,%dx
    ?   1002 jgte .top
    ?   1003 halt
tholmquist@tholmquist:~/Downloads/OS_ch26_hw$
```

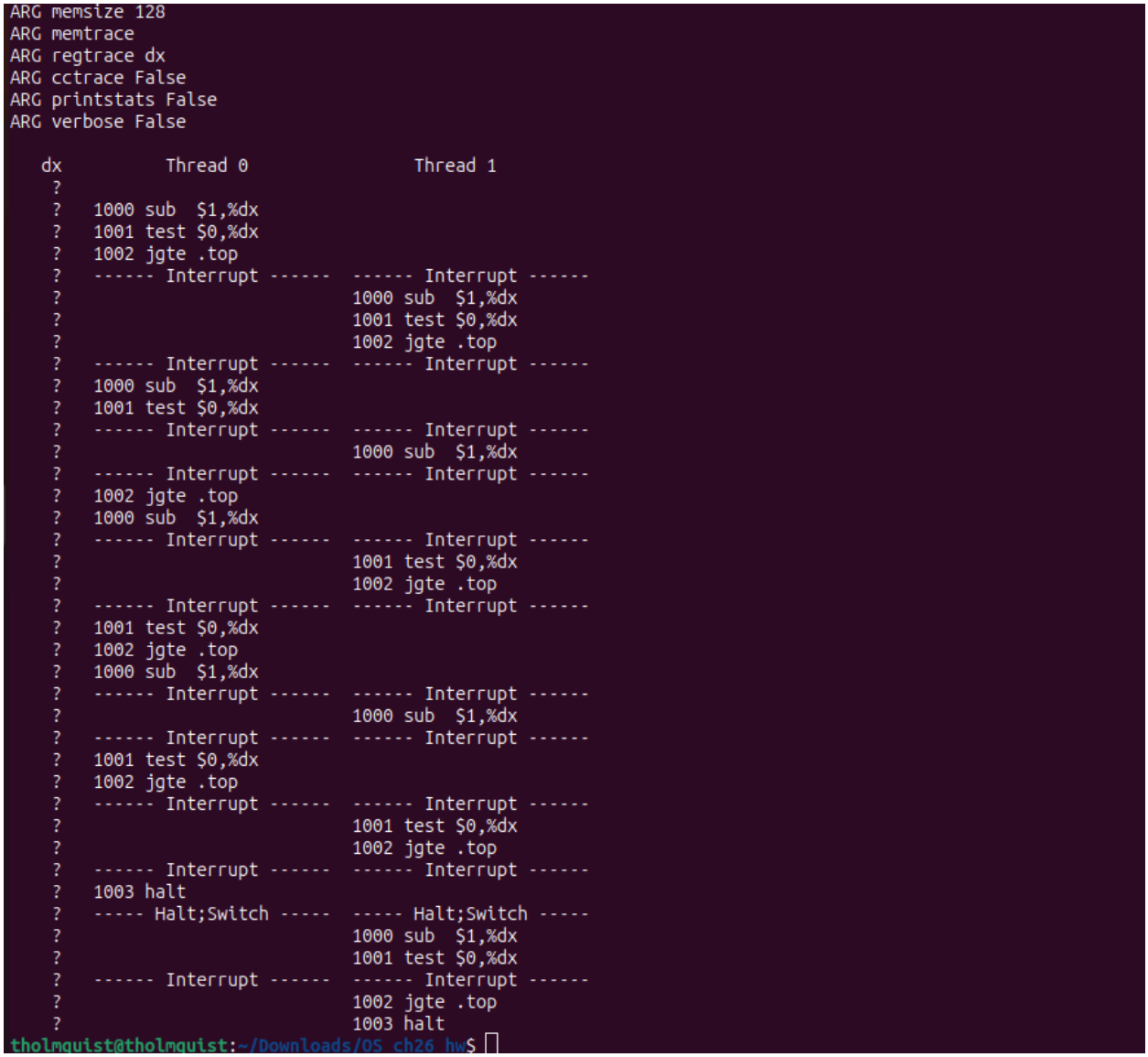
Dx will be 0 to start, then it will be subtracted by 1 making it -1. Since -1 < 0 the test instruction will not be true so the jump instruction will not go and dx will remain -1.

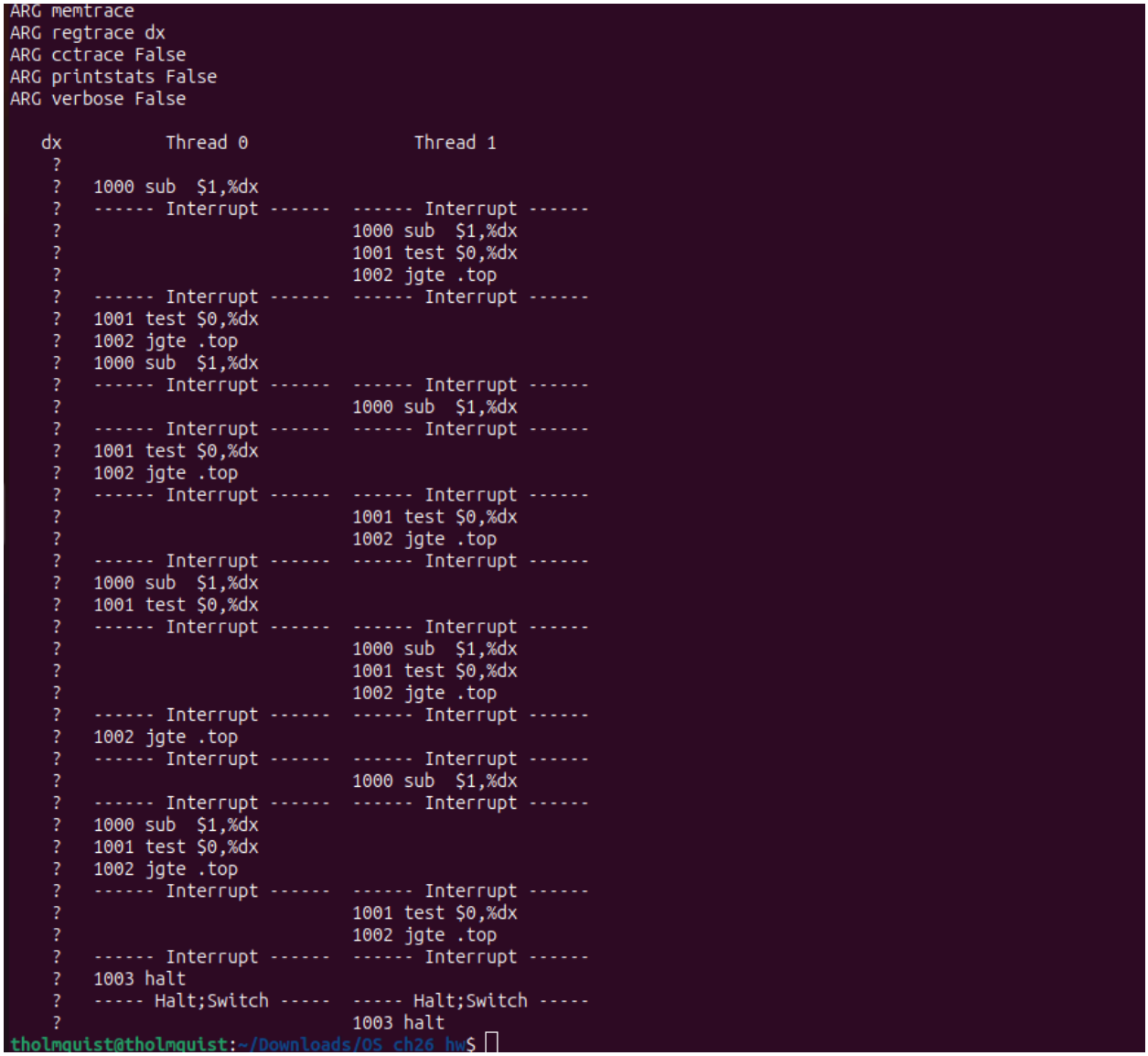
2.



Dx will start at 3, it will then be subtracted repeatedly until it hits -1 because the is less than 0 allowing the loop to break. Then it will move on to the next thread with dx being 3 again. It will then do the same thing subtracting dx by 1 until it hits -1 then ending the last thread. The presence of another thread does affect the calculation and there is no race condition here because we set the interrupt time to be way larger than the time it will take for each thread to complete therefore insuring completion before a race event can occur.

3.


ARG memsize 128
ARG memtrace
ARG regtrace dx
ARG cctrace False
ARG printstats False
ARG verbose False
dx Thread 0 Thread 1
? 1000 sub $1,%dx
? 1001 test $0,%dx
? 1002 jgte .top
? ------ Interrupt ------ ------ Interrupt ------
? 1000 sub $1,%dx
? 1001 test $0,%dx
? 1002 jgte .top
? ------ Interrupt ------ ------ Interrupt ------
? 1000 sub $1,%dx
? 1001 test $0,%dx
? ------ Interrupt ------ ------ Interrupt ------
? 1000 sub $1,%dx
? ------ Interrupt ------ ------ Interrupt ------
? 1002 jgte .top
? 1000 sub $1,%dx
? ------ Interrupt ------ ------ Interrupt ------
? 1001 test $0,%dx
? 1002 jgte .top
? ------ Interrupt ------ ------ Interrupt ------
? 1001 test $0,%dx
? 1002 jgte .top
? 1000 sub $1,%dx
? ------ Interrupt ------ ------ Interrupt ------
? 1000 sub $1,%dx
? ------ Interrupt ------ ------ Interrupt ------
? 1001 test $0,%dx
? 1002 jgte .top
? ------ Interrupt ------ ------ Interrupt ------
? 1001 test $0,%dx
? 1002 jgte .top
? ------ Interrupt ------ ------ Interrupt ------
? 1003 halt
? ----- Halt;Switch ----- ----- Halt;Switch -----
? 1000 sub $1,%dx
? 1001 test $0,%dx
? ------ Interrupt ------ ------ Interrupt ------
? 1002 jgte .top
? 1003 halt
tholmquist@tholmquist:~/Downloads/OS_ch26_hw$



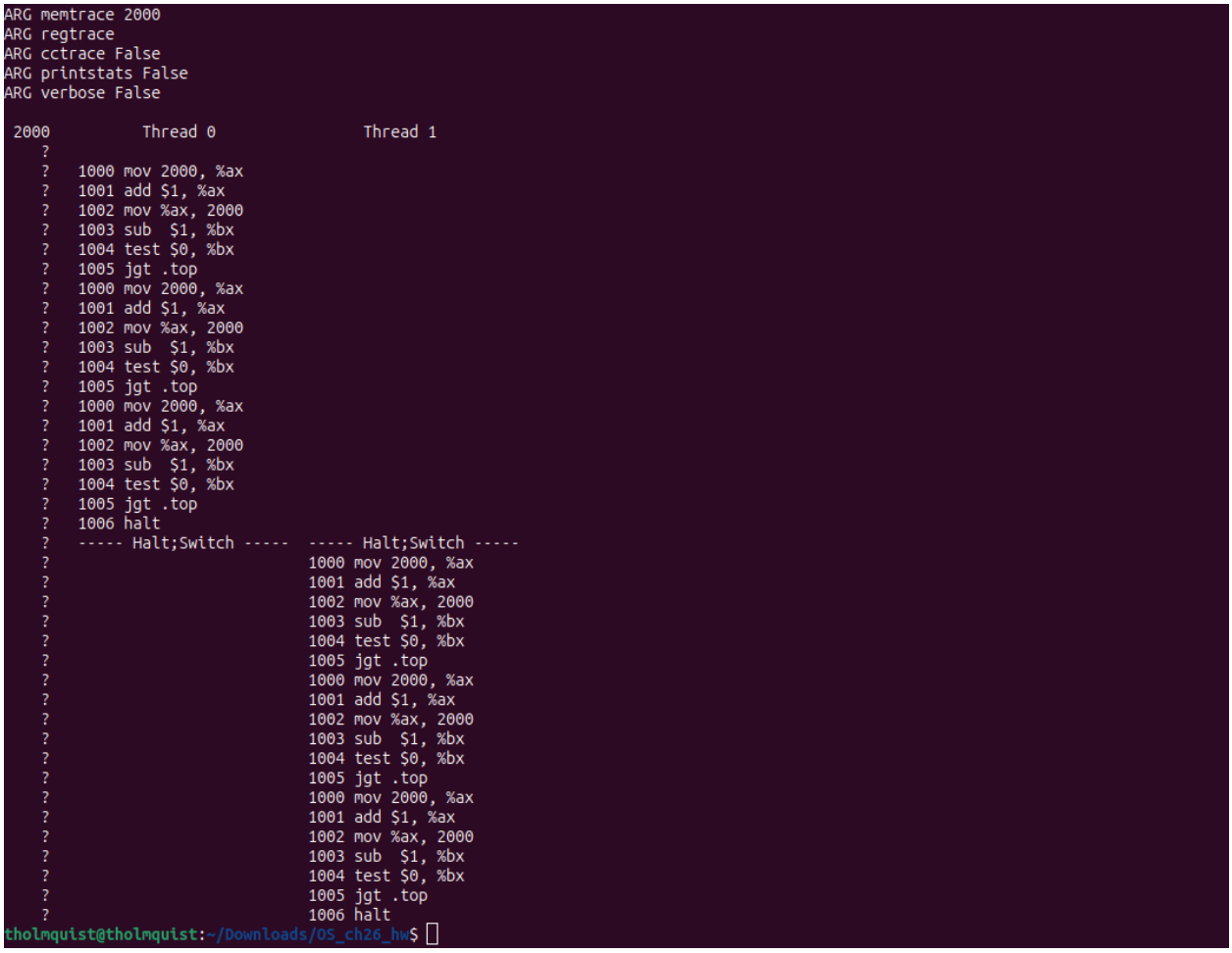
The interrupt frequency does not take away from the output because the two threads are working with different dx inputs and therefore don't affect the outcome of each other if they change the dx register at the wrong time.

4.

```
tholmquist@tholmquist:~/Downloads/OS_ch26_hw$ ./x86.py -p looping-race-nolock.s -t 1 -M 2000
ARG seed 0
ARG numthreads 1
ARG program looping-race-nolock.s
ARG interrupt frequency 50
ARG interrupt randomness False
ARG argv
ARG load address 1000
ARG memsize 128
ARG memtrace 2000
ARG regtrace
ARG cctrace False
ARG printstats False
ARG verbose False

 2000          Thread 0
    ?
    ?   1000 mov 2000, %ax
    ?   1001 add $1, %ax
    ?   1002 mov %ax, 2000
    ?   1003 sub  $1, %bx
    ?   1004 test $0, %bx
    ?   1005 jgt .top
    ?   1006 halt
```

It would start at 0 and then increment by one from the add instruction. Then we subtract 1 from bx but since it is never initialized we don't loop back therefore keeping 2000 at 1.

5.



Each thread loops three times because they both get a bx value of 3 which then decrements until it is 0. The final value at 2000 is 6 because it is incremented from 0 3 times for each thread.

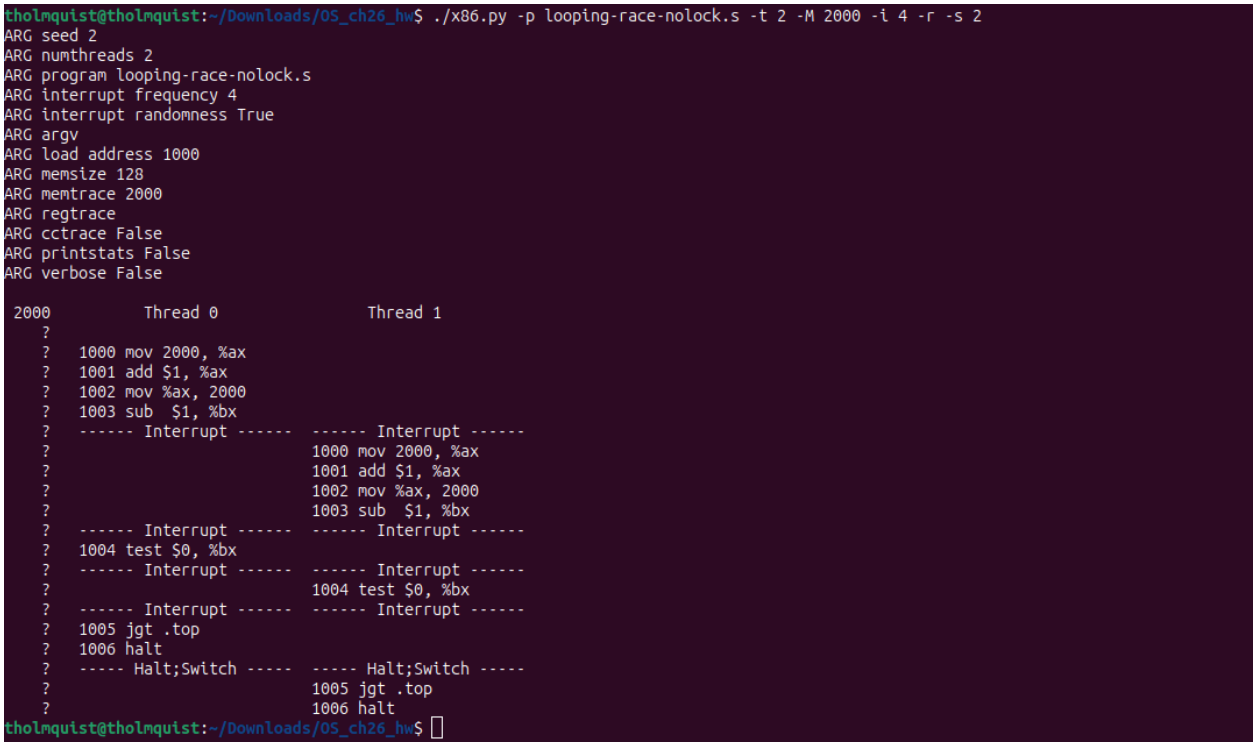
6.

```
tholmquist@tholmquist:~/Downloads/OS_ch26_hw$ ./x86.py -p looping-race-nolock.s -t 2 -M 2000 -i 4 -r -s 0
ARG seed 0
ARG numthreads 2
ARG program looping-race-nolock.s
ARG interrupt frequency 4
ARG interrupt randomness True
ARG argv
ARG load address 1000
ARG memsize 128
ARG memtrace 2000
ARG regtrace
ARG cctrace False
ARG printstats False
ARG verbose False

 2000          Thread 0                Thread 1
    ?
    ?   1000 mov 2000, %ax
    ?   1001 add $1, %ax
    ?   1002 mov %ax, 2000
    ?   1003 sub  $1, %bx
    ?   ------ Interrupt ------  ------ Interrupt ------
    ?                           1000 mov 2000, %ax
    ?                           1001 add $1, %ax
    ?                           1002 mov %ax, 2000
    ?                           1003 sub  $1, %bx
    ?   ------ Interrupt ------  ------ Interrupt ------
    ?   1004 test $0, %bx
    ?   1005 jgt .top
    ?   ------ Interrupt ------  ------ Interrupt ------
    ?                           1004 test $0, %bx
    ?                           1005 jgt .top
    ?   ------ Interrupt ------  ------ Interrupt ------
    ?   1006 halt
    ?   ----- Halt;Switch -----  ----- Halt;Switch -----
    ?                           1006 halt
tholmquist@tholmquist:~/Downloads/OS_ch26_hw$
```

Seed 0

```
ARG seed 1
ARG numthreads 2
ARG program looping-race-nolock.s
ARG interrupt frequency 4
ARG interrupt randomness True
ARG argv
ARG load address 1000
ARG memsize 128
ARG memtrace 2000
ARG regtrace
ARG cctrace False
ARG printstats False
ARG verbose False

 2000          Thread 0                Thread 1
    ?
    ?   1000 mov 2000, %ax
    ?   ------ Interrupt ------  ------ Interrupt ------
    ?                           1000 mov 2000, %ax
    ?                           1001 add $1, %ax
    ?                           1002 mov %ax, 2000
    ?                           1003 sub  $1, %bx
    ?   ------ Interrupt ------  ------ Interrupt ------
    ?   1001 add $1, %ax
    ?   1002 mov %ax, 2000
    ?   1003 sub  $1, %bx
    ?   1004 test $0, %bx
    ?   ------ Interrupt ------  ------ Interrupt ------
    ?                           1004 test $0, %bx
    ?                           1005 jgt .top
    ?   ------ Interrupt ------  ------ Interrupt ------
    ?   1005 jgt .top
    ?   1006 halt
    ?   ----- Halt;Switch -----  ----- Halt;Switch -----
    ?   ------ Interrupt ------  ------ Interrupt ------
    ?                           1006 halt
tholmquist@tholmquist:~/Downloads/OS_ch26_hw$
```

Seed 1



Seed 2

You can tell by looking at the thread interleaving what the final output will be but it's difficult. The timing of the interrupt does matter because if it occurs right after the write call then the value in 2000 will be accurate but if it occurs anytime between that and the read call then it will mess with the values. So therefore the critical section is right after the write call to ensure each thread completes before moving onto the next one.

7.

```
tholmquist@tholmquist:~/Downloads/OS_ch26_hw$ ./x86.py -p looping-race-nolock.s -a bx=1 -t 2 -M 2000 -i 1
ARG seed 0
ARG numthreads 2
ARG program looping-race-nolock.s
ARG interrupt frequency 1
ARG interrupt randomness False
ARG argv bx=1
ARG load address 1000
ARG memsize 128
ARG memtrace 2000
ARG regtrace
ARG cctrace False
ARG printstats False
ARG verbose False

 2000          Thread 0                Thread 1
    ?
    ?   1000 mov 2000, %ax
    ?   ------ Interrupt ------  ------ Interrupt ------
    ?                           1000 mov 2000, %ax
    ?   ------ Interrupt ------  ------ Interrupt ------
    ?   1001 add $1, %ax
    ?   ------ Interrupt ------  ------ Interrupt ------
    ?                           1001 add $1, %ax
    ?   ------ Interrupt ------  ------ Interrupt ------
    ?   1002 mov %ax, 2000
    ?   ------ Interrupt ------  ------ Interrupt ------
    ?                           1002 mov %ax, 2000
    ?   ------ Interrupt ------  ------ Interrupt ------
    ?   1003 sub  $1, %bx
    ?   ------ Interrupt ------  ------ Interrupt ------
    ?                           1003 sub  $1, %bx
    ?   ------ Interrupt ------  ------ Interrupt ------
    ?   1004 test $0, %bx
    ?   ------ Interrupt ------  ------ Interrupt ------
    ?                           1004 test $0, %bx
    ?   ------ Interrupt ------  ------ Interrupt ------
    ?   1005 jgt .top
    ?   ------ Interrupt ------  ------ Interrupt ------
    ?                           1005 jgt .top
    ?   ------ Interrupt ------  ------ Interrupt ------
    ?   1006 halt
    ?   ----- Halt;Switch -----  ----- Halt;Switch -----
    ?   ------ Interrupt ------  ------ Interrupt ------
    ?                           1006 halt
tholmquist@tholmquist:~/Downloads/OS_ch26_hw$
```

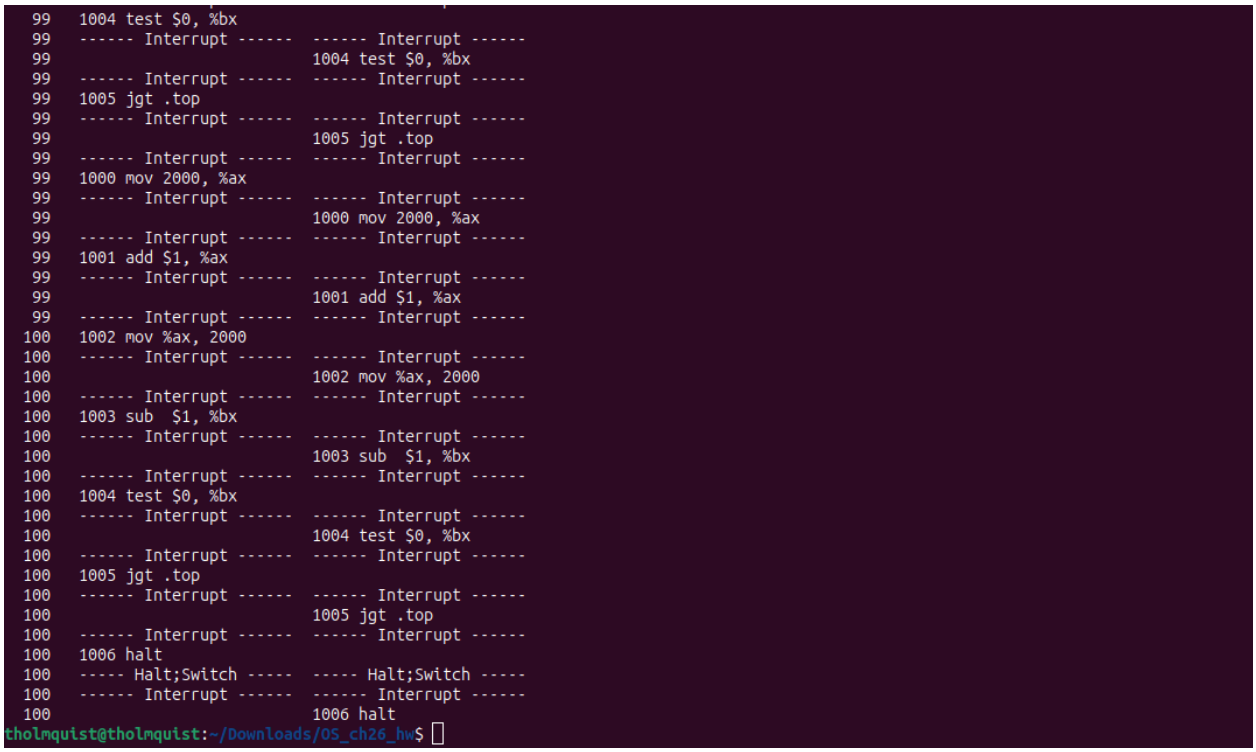
```
tholmquist@tholmquist:~/Downloads/OS_ch26_hw$ ./x86.py -p looping-race-nolock.s -a bx=1 -t 2 -M 2000 -i 2
ARG seed 0
ARG numthreads 2
ARG program looping-race-nolock.s
ARG interrupt frequency 2
ARG interrupt randomness False
ARG argv bx=1
ARG load address 1000
ARG memsize 128
ARG memtrace 2000
ARG regtrace
ARG cctrace False
ARG printstats False
ARG verbose False

 2000          Thread 0                Thread 1
    ?
    ?   1000 mov 2000, %ax
    ?   1001 add $1, %ax
    ?   ------ Interrupt ------  ------ Interrupt ------
    ?                           1000 mov 2000, %ax
    ?                           1001 add $1, %ax
    ?   ------ Interrupt ------  ------ Interrupt ------
    ?   1002 mov %ax, 2000
    ?   1003 sub  $1, %bx
    ?   ------ Interrupt ------  ------ Interrupt ------
    ?                           1002 mov %ax, 2000
    ?                           1003 sub  $1, %bx
    ?   ------ Interrupt ------  ------ Interrupt ------
    ?   1004 test $0, %bx
    ?   1005 jgt .top
    ?   ------ Interrupt ------  ------ Interrupt ------
    ?                           1004 test $0, %bx
    ?                           1005 jgt .top
    ?   ------ Interrupt ------  ------ Interrupt ------
    ?   1006 halt
    ?   ----- Halt;Switch -----  ----- Halt;Switch -----
    ?                           1006 halt
tholmquist@tholmquist:~/Downloads/OS_ch26_hw$
```

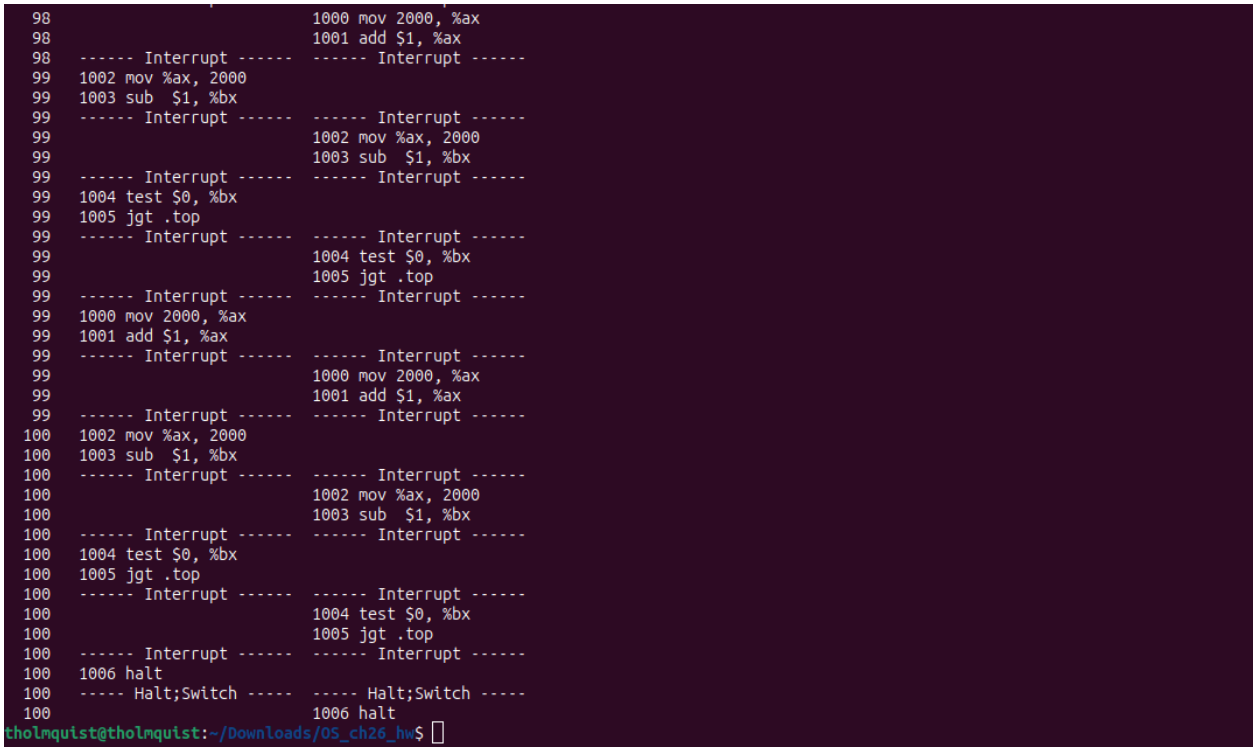
```
tholmquist@tholmquist:~/Downloads/OS_ch26_hw$ ./x86.py -p looping-race-nolock.s -a bx=1 -t 2 -M 2000 -i 3
ARG seed 0
ARG numthreads 2
ARG program looping-race-nolock.s
ARG interrupt frequency 3
ARG interrupt randomness False
ARG argv bx=1
ARG load address 1000
ARG memsize 128
ARG memtrace 2000
ARG regtrace
ARG cctrace False
ARG printstats False
ARG verbose False

 2000          Thread 0                Thread 1
    ?
    ?   1000 mov 2000, %ax
    ?   1001 add $1, %ax
    ?   1002 mov %ax, 2000
    ?   ------ Interrupt ------  ------ Interrupt ------
    ?                           1000 mov 2000, %ax
    ?                           1001 add $1, %ax
    ?                           1002 mov %ax, 2000
    ?   ------ Interrupt ------  ------ Interrupt ------
    ?   1003 sub  $1, %bx
    ?   1004 test $0, %bx
    ?   1005 jgt .top
    ?   ------ Interrupt ------  ------ Interrupt ------
    ?                           1003 sub  $1, %bx
    ?                           1004 test $0, %bx
    ?                           1005 jgt .top
    ?   ------ Interrupt ------  ------ Interrupt ------
    ?   1006 halt
    ?   ----- Halt;Switch -----  ----- Halt;Switch -----
    ?                           1006 halt
tholmquist@tholmquist:~/Downloads/OS_ch26_hw$
```

For 1 the final value will be 1 because both threads will save the value 0 and increment that to 1 saving the same value 1 back into 2000 therefore not incrementing it correctly. 2 will do the same thing and 3 will finally give enough time between interrupts for thread

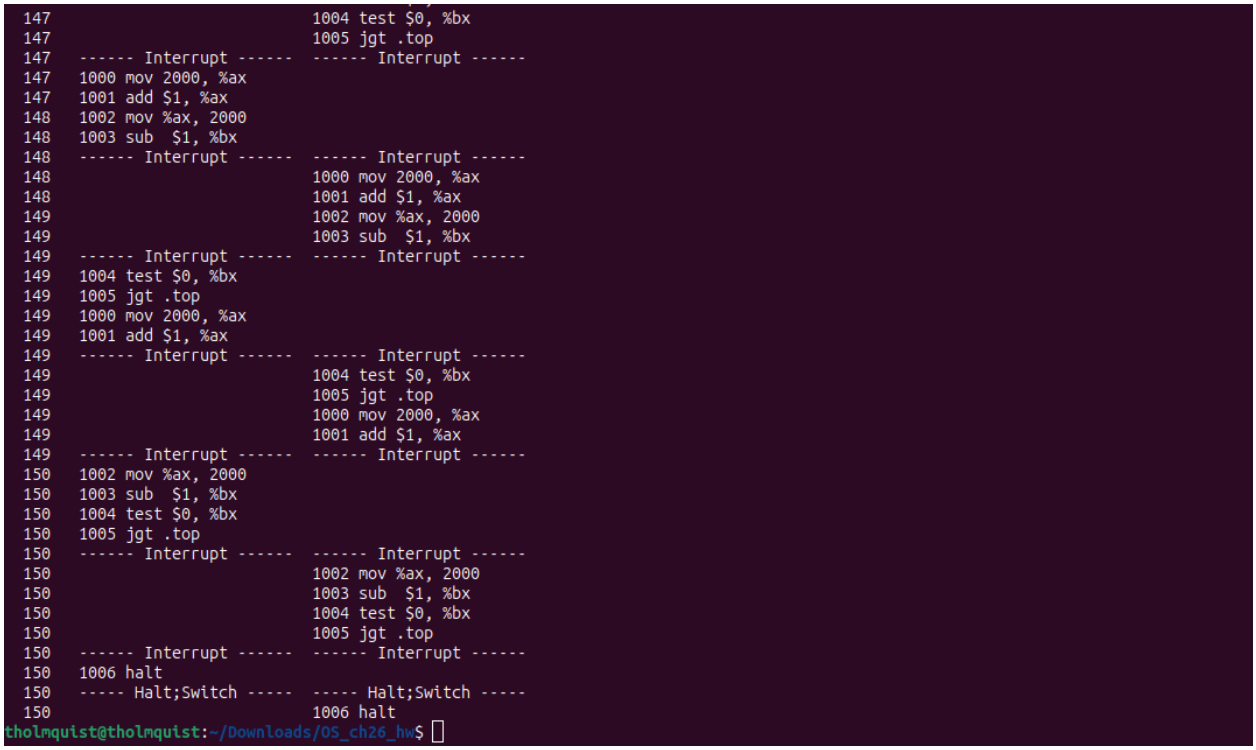
0 to complete and store the new value back before thread 1 begins, allowing it to increment the correct value.

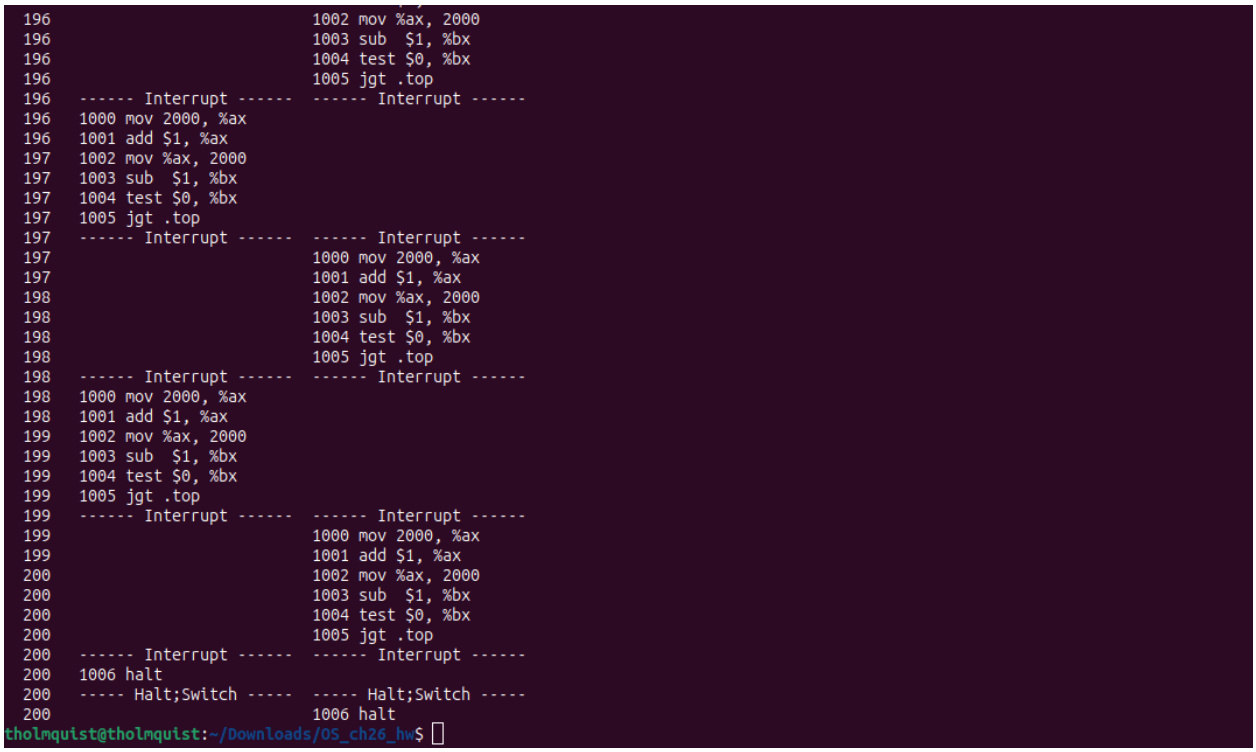
8.





```
 196   ------ Interrupt ------  ------ Interrupt ------
 196   1000 mov 2000, %ax
 196   1001 add $1, %ax
 197   1002 mov %ax, 2000
 197   ------ Interrupt ------  ------ Interrupt ------
 197                            1000 mov 2000, %ax
 197                            1001 add $1, %ax
 198                            1002 mov %ax, 2000
 198   ------ Interrupt ------  ------ Interrupt ------
 198   1003 sub  $1, %bx
 198   1004 test $0, %bx
 198   1005 jgt .top
 198   ------ Interrupt ------  ------ Interrupt ------
 198                            1003 sub  $1, %bx
 198                            1004 test $0, %bx
 198                            1005 jgt .top
 198   ------ Interrupt ------  ------ Interrupt ------
 198   1000 mov 2000, %ax
 198   1001 add $1, %ax
 199   1002 mov %ax, 2000
 199   ------ Interrupt ------  ------ Interrupt ------
 199                            1000 mov 2000, %ax
 199                            1001 add $1, %ax
 200                            1002 mov %ax, 2000
 200   ------ Interrupt ------  ------ Interrupt ------
 200   1003 sub  $1, %bx
 200   1004 test $0, %bx
 200   1005 jgt .top
 200   ------ Interrupt ------  ------ Interrupt ------
 200                            1003 sub  $1, %bx
 200                            1004 test $0, %bx
 200                            1005 jgt .top
 200   ------ Interrupt ------  ------ Interrupt ------
 200   1006 halt
 200   ----- Halt;Switch -----  ----- Halt;Switch -----
 200                            1006 halt
tholmquist@tholmquist:~/Downloads/OS_ch26_hw$
```





As seen here with the examples of intervals 1,2,3,4,and 6 if you have intervals that are divisible by 3 then it gives time for the entire load, add and store calls to complete whereas any other numbers will stop the process early and mess with the final outcome. The interval that surprised me initially was 4 because I had assumed that anything

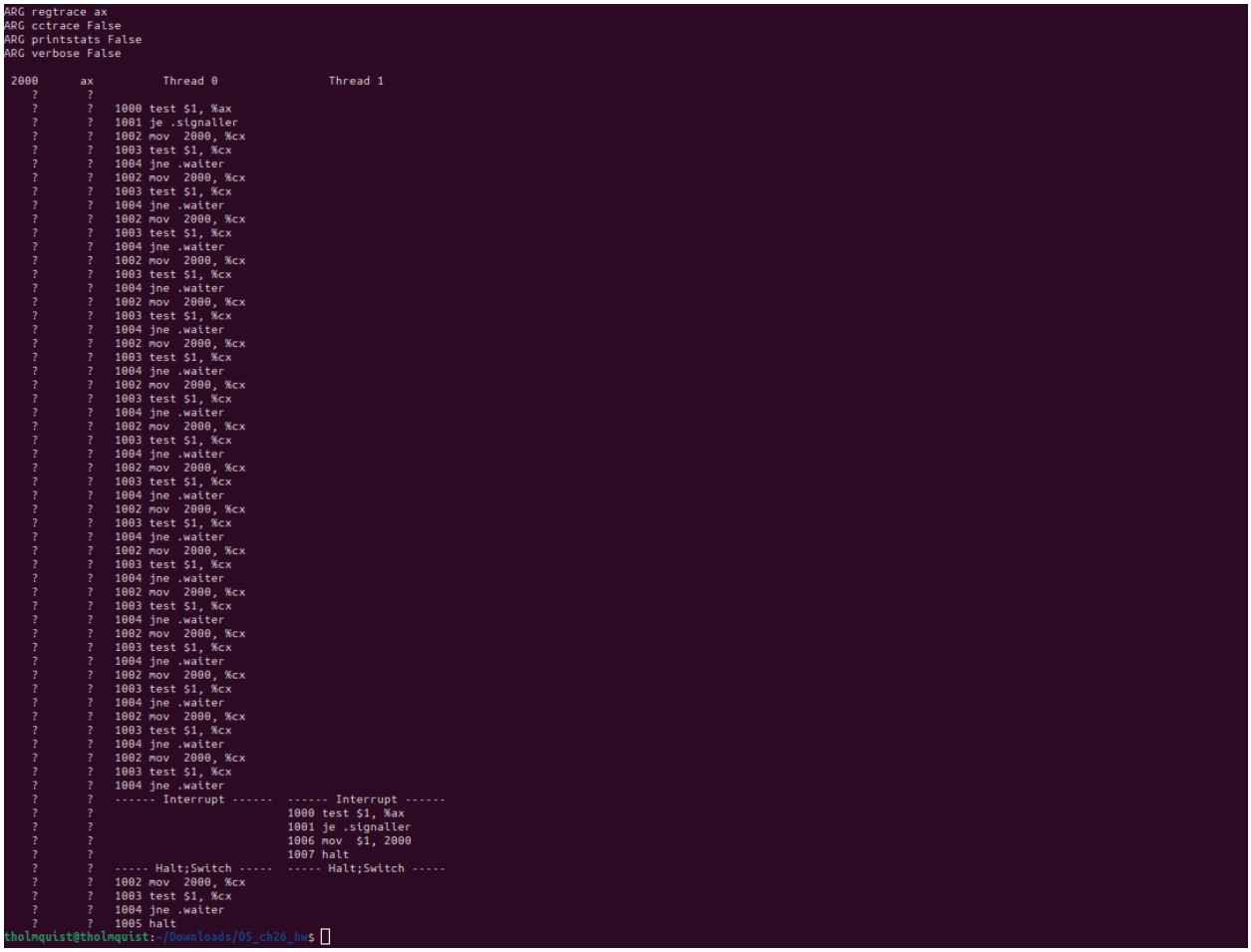
greater than three would be accurate but then it quickly made sense that it must be intervals of three in order to not stop the process early.

9.

```
tholmquist@tholmquist:~/Downloads/OS_ch26_hw$ ./x86.py -p wait-for-me.s -a ax=1,ax=0 -R ax -M 2000
ARG seed 0
ARG numthreads 2
ARG program wait-for-me.s
ARG interrupt frequency 50
ARG interrupt randomness False
ARG argv ax=1,ax=0
ARG load address 1000
ARG memsize 128
ARG memtrace 2000
ARG regtrace ax
ARG cctrace False
ARG printstats False
ARG verbose False

 2000      ax          Thread 0                Thread 1
    ?       ?
    ?       ?   1000 test $1, %ax
    ?       ?   1001 je .signaller
    ?       ?   1006 mov  $1, 2000
    ?       ?   1007 halt
    ?       ?   ----- Halt;Switch -----  ----- Halt;Switch -----
    ?       ?                           1000 test $1, %ax
    ?       ?                           1001 je .signaller
    ?       ?                           1002 mov  2000, %cx
    ?       ?                           1003 test $1, %cx
    ?       ?                           1004 jne .waiter
    ?       ?                           1005 halt
tholmquist@tholmquist:~/Downloads/OS_ch26_hw$
```

The code should behave such that Thread 0 writes the value 1 to 2000 and then halts, while Thread 1 checks the value of ax and proceeds without altering 2000. As a result, the final value at memory location 2000 will be 1. The value at 2000 is being used by the threads to let eachother know what execution they should do based on what the other one has done already.

10.



The threads behave the same but reversed. Thread 0 is waiting for a 1 in the memory location 2000 until it gets interrupted. More random or frequent interrupt intervals would help this program so that thread 0 doesn't have to take so much time waiting on 2000 to change. This program is not efficiently using the cpu because thread 0 is taking up a great amount of time waiting on a process that has to wait for thread 0 to finish before it can start.