```
mov 2000, %ax    load from address 2000 and store in %ax
mov %ax, 2000    store value of address 2000 in %ax
add $1, %ax      adds integer 1 to value of %ax
sub $1,%dx       subtracts integer 1 from value of %ax
halt             stops the running thread
.main .top       jump flags
test $0,%dx      tests %dx to be '>=' '>' '<=' '<' '!=' '==' integer 0
jgte .top        jump if 2nd value of test is greater than or equal to first value of test

General purpose registers: %ax, %bx, %cx, %dx
```

Question 1

```
Python3 ./x86.py -p loop.s -t 1 -i 100 -R dx
```
This specifies a single thread, an interrupt every 100 instructions, trace of register %dx. What will %dx be during the run?

```
   dx          Thread 0
    0
   -1   1000 sub  $1,%dx
   -1   1001 test $0,%dx
   -1   1002 jgte .top
   -1   1003 halt
```

Question 2

```
Python3 ./x86.py -p loop.s -t 2 -i 100 -a dx=3,dx=3 -R dx
```
This specifies two threads, and initializes each %dx to 3. What values will %dx see?
Does the presence of multiple threads affect your calculations?
Is there a race in this code?

```
.main
.top
sub  $1,%dx
test $0,%dx
jgte .top
halt
```

```
   dx          Thread 0                Thread 1
    3
    2   1000 sub  $1,%dx
    2   1001 test $0,%dx
    2   1002 jgte .top
    1   1000 sub  $1,%dx
    1   1001 test $0,%dx
    1   1002 jgte .top
    0   1000 sub  $1,%dx
    0   1001 test $0,%dx
    0   1002 jgte .top
   -1   1000 sub  $1,%dx
   -1   1001 test $0,%dx
   -1   1002 jgte .top
   -1   1003 halt
    3   ----- Halt;Switch -----  ----- Halt;Switch -----
    2                             1000 sub  $1,%dx
    2                             1001 test $0,%dx
    2                             1002 jgte .top
    1                             1000 sub  $1,%dx
    1                             1001 test $0,%dx
    1                             1002 jgte .top
    0                             1000 sub  $1,%dx
    0                             1001 test $0,%dx
    0                             1002 jgte .top
   -1                             1000 sub  $1,%dx
   -1                             1001 test $0,%dx
   -1                             1002 jgte .top
   -1                             1003 halt
```

Since we don't have any loads from memory we don't encounter any race conditions

Question 3

```
Python3 ./x86.py -p loop.s -t 2 -i 3 -r -a dx=3,dx=3 -R dx
```
This makes the interrupt interval small/random; use different seeds (-s) to see different interleavings.
Does the interrupt frequency change anything?

```
.main
.top
sub  $1,%dx
test $0,%dx
jgte .top
halt
```

```
   dx          Thread 0                Thread 1
    3
    2   1000 sub  $1,%dx
    2   1001 test $0,%dx
    2   1002 jgte .top
    3   ------ Interrupt ------  ------ Interrupt ------
    2                             1000 sub  $1,%dx
    2                             1001 test $0,%dx
    2                             1002 jgte .top
    2   ------ Interrupt ------  ------ Interrupt ------
    1   1000 sub  $1,%dx
    1   1001 test $0,%dx
    2   ------ Interrupt ------  ------ Interrupt ------
    1                             1000 sub  $1,%dx
    1   ------ Interrupt ------  ------ Interrupt ------
    1   1002 jgte .top
    0   1000 sub  $1,%dx
    1   ------ Interrupt ------  ------ Interrupt ------
    1                             1001 test $0,%dx
    1                             1002 jgte .top
    0   ------ Interrupt ------  ------ Interrupt ------
    0   1001 test $0,%dx
    0   1002 jgte .top
   -1   1000 sub  $1,%dx
    1   ------ Interrupt ------  ------ Interrupt ------
    0                             1000 sub  $1,%dx
   -1   ------ Interrupt ------  ------ Interrupt ------
   -1   1001 test $0,%dx
   -1   1002 jgte .top
    0   ------ Interrupt ------  ------ Interrupt ------
    0                             1001 test $0,%dx
    0                             1002 jgte .top
   -1   ------ Interrupt ------  ------ Interrupt ------
   -1   1003 halt
    0   ----- Halt;Switch -----  ----- Halt;Switch -----
   -1                             1000 sub  $1,%dx
   -1                             1001 test $0,%dx
   -1   ------ Interrupt ------  ------ Interrupt ------
   -1                             1002 jgte .top
   -1                             1003 halt
```

The interrupt doesn't change anything, since every threads has its own registers and we still don't load anything from memory.

Question 4

Now, a different program, looping-race-nolock.s, which accesses a shared variable located at address 2000; we'll call this variable value.
Run it with a single thread to confirm your under- standing:
```
Python3 ./x86.py -p looping-race-nolock.s -t 1 -M 2000
```
What is value (i.e., at memory address 2000) throughout the run?

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

```
# assumes %bx has loop count in it
.main
.top
# critical section
mov 2000, %ax  # get 'value' at address 2000
add $1, %ax    # increment it
mov %ax, 2000  # store it back
# see if we're still looping
sub  $1, %bx
test $0, %bx
jgt .top
halt
```

Question 5

Run with multiple iterations/threads:
```
Python3 ./x86.py -p looping-race-nolock.s -t 2 -a bx=3 -M 2000
```
Why does each thread loop three times? What is final value of value?

```
 2000          Thread 0                Thread 1
    0
    0   1000 mov 2000, %ax
    0   1001 add $1, %ax
    1   1002 mov %ax, 2000
    1   1003 sub  $1, %bx
    1   1004 test $0, %bx
    1   1005 jgt .top
    1   1000 mov 2000, %ax
    1   1001 add $1, %ax
    2   1002 mov %ax, 2000
    2   1003 sub  $1, %bx
    2   1004 test $0, %bx
    2   1005 jgt .top
    2   1000 mov 2000, %ax
    3   1001 add $1, %ax
    3   1002 mov %ax, 2000
    3   1003 sub  $1, %bx
    3   1004 test $0, %bx
    3   1005 jgt .top
    3   1006 halt
    3   ----- Halt;Switch -----  ----- Halt;Switch -----
    3                             1000 mov 2000, %ax
    3                             1001 add $1, %ax
    4                             1002 mov %ax, 2000
    4                             1003 sub  $1, %bx
    4                             1004 test $0, %bx
    4                             1005 jgt .top
    4                             1000 mov 2000, %ax
    4                             1001 add $1, %ax
    5                             1002 mov %ax, 2000
    5                             1003 sub  $1, %bx
    5                             1004 test $0, %bx
    5                             1005 jgt .top
    5                             1000 mov 2000, %ax
    5                             1001 add $1, %ax
    6                             1002 mov %ax, 2000
    6                             1003 sub  $1, %bx
    6                             1004 test $0, %bx
    6                             1005 jgt .top
    6                             1006 halt
```

Each thread loops 3 times because of the %bx register that keeps track of the loop count and is initialised with 3. Because of the really high interrupt frequency each thread can run to completion and thus no race condition can occur.

Question 6

Run with random interrupt intervals:
```
Python3 ./x86.py -p looping-race-nolock.s -t 2 -M 2000 -i 4 -r -s 0
```
with different seeds (-s 1, -s 2, etc.) Can you tell by looking at the thread interleaving what the final value of value will be?
Does the timing of the interrupt matter?
Where can it safely occur and Where not?
In other words, where is the critical section exactly?

```
 2000          Thread 0                Thread 1
    0
    0    1000 mov 2000, %ax
    0    1001 add $1, %ax
    1    1002 mov %ax, 2000
    1    1003 sub  $1, %bx
    1    ------ Interrupt ------  ------ Interrupt ------
    1                              1000 mov 2000, %ax
    1                              1001 add $1, %ax
    2                              1002 mov %ax, 2000
    2                              1003 sub  $1, %bx
    2    ------ Interrupt ------  ------ Interrupt ------
    2    1004 test $0, %bx
    2    1005 jgt .top
    2    ------ Interrupt ------  ------ Interrupt ------
    2                              1004 test $0, %bx
    2                              1005 jgt .top
    2    ------ Interrupt ------  ------ Interrupt ------
    2    1006 halt
    2    ----- Halt;Switch -----  ----- Halt;Switch -----
    2                              1006 halt
```

An interrupt can safely occur when the critical section is run completely

```
# critical section
mov 2000, %ax  # get 'value' at address 2000
add $1, %ax    # increment it
mov %ax, 2000  # store it back
```

Question 7

Now examine fixed interrupt intervals:
```
Python3 ./x86.py -p looping-race-nolock.s -a bx=1 -t 2 -M 2000 -i 1
```
What will the final value of the shared variable value be?
What about when you change -i 2, -i 3, etc.?
For which interrupt intervals does the program give the “correct” answer?

```
ARG verbose False
```

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

When the interrupt interval is at least as big as the critical section we will get the correct result
In this case: **-i >= 3**

Question 8

```
Python3 ./x86.py -p looping-race-nolock.s -a bx=100 -t 2 -M 2000 -i 3
```
Run the same for more loops (e.g., set -a bx=100). What interrupt intervals (-i) lead to a correct outcome?
Which intervals are surprising?

As long as the interrupt interval is >= 3 the loop count set by %bx is not relevant. However, we will always get at least as many loops as there are threads.

Question 9

One last program: wait-for-me.s. Run:
```
Python3 ./x86.py -p wait-for-me.s -a ax=1,ax=0 -R ax -M 2000
```
This sets the %ax register to 1 for thread 0, and 0 for thread 1, and watches %ax and memory location 2000.
How should the code behave?
How is the value at location 2000 being used by the threads?
What will its final value be?

```
 2000        ax          Thread 0                Thread 1
    0         1
    0         1    1000 test $1, %ax
    0         1    1001 je .signaller
    1         1    1006 mov  $1, 2000
    1         1    1007 halt
    1         0    ----- Halt;Switch -----    ----- Halt;Switch -----
    1         0                               1000 test $1, %ax
    1         0                               1001 je .signaller
    1         0                               1002 mov  2000, %cx
    1         0                               1003 test $1, %cx
    1         0                               1004 jne .waiter
    1         0                               1005 halt
```

Question 10

Now switch the inputs:
```
Python3 ./x86.py -p wait-for-me.s -a ax=0,ax=1 -R ax -M 2000
```
How do the threads behave? What is thread 0 doing?
How would changing the interrupt interval (e.g., -i 1000, or perhaps to use random intervals) change the trace outcome?
Is the program efficiently using the CPU?

```
 2000      ax          Thread 0                Thread 1
    0       0
    0       0   1000 test $1, %ax
    0       0   1001 je .signaller
    0       0   1002 mov  2000, %cx
    0       0   1003 test $1, %cx
    0       0   1004 jne .waiter
    0       0   1002 mov  2000, %cx
    0       0   1003 test $1, %cx
    0       0   1004 jne .waiter
    0       0   1002 mov  2000, %cx
    0       0   1003 test $1, %cx
    0       0   1004 jne .waiter
    0       0   1002 mov  2000, %cx
    0       0   1003 test $1, %cx
    0       0   1004 jne .waiter
    0       0   1002 mov  2000, %cx
    0       0   1003 test $1, %cx
    0       0   1004 jne .waiter
    0       0   1002 mov  2000, %cx
    0       0   1003 test $1, %cx
    0       0   1004 jne .waiter
    0       0   1002 mov  2000, %cx
    0       0   1003 test $1, %cx
    0       0   1004 jne .waiter
    0       0   1002 mov  2000, %cx
    0       0   1003 test $1, %cx
    0       0   1004 jne .waiter
    0       0   1002 mov  2000, %cx
    0       0   1003 test $1, %cx
    0       0   1004 jne .waiter
    0       0   1002 mov  2000, %cx
    0       0   1003 test $1, %cx
    0       0   1004 jne .waiter
    0       0   1002 mov  2000, %cx
    0       0   1003 test $1, %cx
    0       0   1004 jne .waiter
    0       0   1002 mov  2000, %cx
    0       0   1003 test $1, %cx
    0       0   1004 jne .waiter
    0       0   1002 mov  2000, %cx
    0       0   1003 test $1, %cx
    0       0   1004 jne .waiter
    0       0   1002 mov  2000, %cx
    0       0   1003 test $1, %cx
    0       0   1004 jne .waiter
    0       0   1002 mov  2000, %cx
    0       0   1003 test $1, %cx
    0       0   1004 jne .waiter
    0       0   1002 mov  2000, %cx
    0       0   1003 test $1, %cx
    0       0   1004 jne .waiter
    0       0   1002 mov  2000, %cx
    0       0   1003 test $1, %cx
    0       0   1004 jne .waiter
    0       0   1002 mov  2000, %cx
    0       0   1003 test $1, %cx
    0       0   1004 jne .waiter
    0       0   1002 mov  2000, %cx
    0       0   1003 test $1, %cx
    0       0   1004 jne .waiter
    0       0   1002 mov  2000, %cx
    0       0   1003 test $1, %cx
    0       0   1004 jne .waiter
    0       0   1002 mov  2000, %cx
    0       0   1003 test $1, %cx
    0       0   1004 jne .waiter
    0       0   1002 mov  2000, %cx
    0       0   1003 test $1, %cx
    0       0   1004 jne .waiter
    0       0   1002 mov  2000, %cx
    0       0   1003 test $1, %cx
    0       0   1004 jne .waiter
    0       0   1002 mov  2000, %cx
    0       0   1003 test $1, %cx
    0       0   1004 jne .waiter
    0       0   1002 mov  2000, %cx
    0       0   1003 test $1, %cx
    0       0   1004 jne .waiter
    0       0   1002 mov  2000, %cx
    0       0   1003 test $1, %cx
    0       0   1004 jne .waiter
    0       0   1002 mov  2000, %cx
    0       0   1003 test $1, %cx
    0       0   1004 jne .waiter
    0       0   1002 mov  2000, %cx
    0       0   1003 test $1, %cx
    0       0   1004 jne .waiter
    0       1   ------ Interrupt ------  ------ Interrupt ------
    0       1                            1000 test $1, %ax
    0       1                            1001 je .signaller
    1       1                            1006 mov  $1, 2000
    1       1                            1007 halt
    1       0   ----- Halt;Switch -----  ----- Halt;Switch -----
    1       0   1002 mov  2000, %cx
    1       0   1003 test $1, %cx
    1       0   1004 jne .waiter
    1       0   1005 halt
```

Thread 0 just waits until thread 1 sets the %ax value to 1. This is really inefficient since we need to wait until an interrupt occurs.