

```
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
jgte $0, %dx        tests %dx to be '>=' '>' '<=' '<' '!=' '==' integer 0
jgte .top          jumps 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?
```

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

```
dx      Thread 0      Thread 1      .main
3      1000 sub $1, %dx      1000 sub $1, %dx      .top
2      1001 test $0, %dx      1001 test $0, %dx      sub $1, %dx
2      1002 jgte .top          1002 jgte .top          test $0, %dx
3      ----- Interrupt -----      1000 sub $1, %dx      jgte .top
2      1000 sub $1, %dx      1001 test $0, %dx      halt
2      1001 test $0, %dx      1002 jgte .top
2      ----- Interrupt -----      ----- Interrupt -----
1      1000 sub $1, %dx      1000 sub $1, %dx
1      1001 test $0, %dx      1001 test $0, %dx
2      ----- Interrupt -----      ----- Interrupt -----
1      1000 sub $1, %dx      1000 sub $1, %dx
1      1002 jgte .top          1002 jgte .top
0      1000 sub $1, %dx      1002 jgte .top
1      ----- Interrupt -----      ----- Interrupt -----
1      1001 test $0, %dx      1001 test $0, %dx
1      1002 jgte .top          1002 jgte .top
0      ----- Interrupt -----      ----- Interrupt -----
0      1001 test $0, %dx      1002 jgte .top
0      1002 jgte .top          1000 sub $1, %dx
-1     1000 sub $1, %dx      ----- Interrupt -----
1      ----- Interrupt -----      1000 sub $1, %dx
0      1001 test $0, %dx      ----- Interrupt -----
0      1002 jgte .top          1001 test $0, %dx
-1     1000 sub $1, %dx      1002 jgte .top
1      ----- Interrupt -----      ----- Interrupt -----
0      1001 test $0, %dx      1002 jgte .top
0      1002 jgte .top          1000 sub $1, %dx
-1     1000 sub $1, %dx      ----- Interrupt -----
1      ----- Interrupt -----      1000 sub $1, %dx
0      1001 test $0, %dx      ----- Interrupt -----
0      1002 jgte .top          1001 test $0, %dx
-1     1000 sub $1, %dx      1002 jgte .top
1      ----- Interrupt -----      1003 halt
0      1001 test $0, %dx      1003 halt
-1     1002 jgte .top          1003 halt
-1     1002 jgte .top          1003 halt
0      ----- Interrupt -----      1003 halt
0      1001 test $0, %dx      1003 halt
-1     1002 jgte .top          1003 halt
-1     1002 jgte .top          1003 halt
0      ----- Interrupt -----      1003 halt
0      1001 test $0, %dx      1003 halt
-1     1002 jgte .top          1003 halt
-1     1002 jgte .top          1003 halt
0      ----- Interrupt -----      1003 halt
0      1001 test $0, %dx      1003 halt
-1     1002 jgte .top          1003 halt
-1     1002 jgte .top          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 understanding:

```
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      # assumes %bx has loop count in it
0      .main
0      1000 mov 2000, %ax      .top
0      1001 add $1, %ax      # critical section
1      1002 mov %ax, 2000      mov 2000, %ax # get 'value' at address 2000
1      1003 sub $1, %bx      add $1, %ax # increment it
1      1004 test $0, %bx      mov %ax, 2000 # store it back
1      1005 jgt .top          # see if we're still looping
1      1006 halt              sub $1, %bx
1      1006 halt              test $0, %bx
1      1006 halt              jgt .top
1      1006 halt              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      1000 mov 2000, %ax      1000 mov 2000, %ax
0      1001 add $1, %ax      1001 add $1, %ax
0      1002 mov %ax, 2000      1002 mov %ax, 2000
1      1003 sub $1, %bx      1003 sub $1, %bx
1      1004 test $0, %bx      1004 test $0, %bx
1      1005 jgt .top          1005 jgt .top
1      1000 mov 2000, %ax      1000 mov 2000, %ax
1      1001 add $1, %ax      1001 add $1, %ax
2      1002 mov %ax, 2000      1002 mov %ax, 2000
2      1003 sub $1, %bx      1003 sub $1, %bx
2      1004 test $0, %bx      1004 test $0, %bx
2      1005 jgt .top          1005 jgt .top
2      1000 mov 2000, %ax      1000 mov 2000, %ax
3      1001 add $1, %ax      1001 add $1, %ax
3      1002 mov %ax, 2000      1002 mov %ax, 2000
3      1003 sub $1, %bx      1003 sub $1, %bx
3      1004 test $0, %bx      1004 test $0, %bx
3      1005 jgt .top          1005 jgt .top
3      1006 halt              1006 halt
3      ----- Halt;Switch -----      ----- Halt;Switch -----
3      1000 mov 2000, %ax      1000 mov 2000, %ax
3      1001 add $1, %ax      1001 add $1, %ax
4      1002 mov %ax, 2000      1002 mov %ax, 2000
4      1003 sub $1, %bx      1003 sub $1, %bx
4      1004 test $0, %bx      1004 test $0, %bx
4      1005 jgt .top          1005 jgt .top
4      1000 mov 2000, %ax      1000 mov 2000, %ax
4      1001 add $1, %ax      1001 add $1, %ax
5      1002 mov %ax, 2000      1002 mov %ax, 2000
5      1003 sub $1, %bx      1003 sub $1, %bx
5      1004 test $0, %bx      1004 test $0, %bx
5      1005 jgt .top          1005 jgt .top
5      1000 mov 2000, %ax      1000 mov 2000, %ax
5      1001 add $1, %ax      1001 add $1, %ax
6      1002 mov %ax, 2000      1002 mov %ax, 2000
6      1003 sub $1, %bx      1003 sub $1, %bx
6      1004 test $0, %bx      1004 test $0, %bx
6      1005 jgt .top          1005 jgt .top
6      1006 halt              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      1000 mov 2000, %ax      1000 mov 2000, %ax
0      1001 add $1, %ax      1001 add $1, %ax
0      1002 mov %ax, 2000      1002 mov %ax, 2000
1      1003 sub $1, %bx      1003 sub $1, %bx
1      ----- Interrupt -----      ----- Interrupt -----
1      1000 mov 2000, %ax      1000 mov 2000, %ax
1      1001 add $1, %ax      1001 add $1, %ax
2      1002 mov %ax, 2000      1002 mov %ax, 2000
2      1003 sub $1, %bx      1003 sub $1, %bx
2      ----- Interrupt -----      ----- Interrupt -----
2      1004 test $0, %bx      1004 test $0, %bx
2      1005 jgt .top          1005 jgt .top
2      ----- Interrupt -----      ----- Interrupt -----
2      1000 mov 2000, %ax      1000 mov 2000, %ax
2      1001 add $1, %ax      1001 add $1, %ax
2      1002 mov %ax, 2000      1002 mov %ax, 2000
2      1003 sub $1, %bx      1003 sub $1, %bx
2      ----- Interrupt -----      ----- Interrupt -----
2      1004 test $0, %bx      1004 test $0, %bx
2      1005 jgt .top          1005 jgt .top
2      ----- Interrupt -----      ----- Interrupt -----
2      1006 halt              1006 halt
2      ----- Halt;Switch -----      ----- Halt;Switch -----
2      1000 mov 2000, %ax      1000 mov 2000, %ax
2      1001 add $1, %ax      1001 add $1, %ax
2      1002 mov %ax, 2000      1002 mov %ax, 2000
2      1003 sub $1, %bx      1003 sub $1, %bx
2      ----- Interrupt -----      ----- Interrupt -----
2      1004 test $0, %bx      1004 test $0, %bx
2      1005 jgt .top          1005 jgt .top
2      ----- Interrupt -----      ----- Interrupt -----
2      1006 halt              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      1000 mov 2000, %ax      1000 mov 2000, %ax
0      ----- Interrupt -----      ----- Interrupt -----
0      1000 mov 2000, %ax      1000 mov 2000, %ax
0      ----- Interrupt -----      ----- Interrupt -----
0      1001 add $1, %ax      1001 add $1, %ax
0      ----- Interrupt -----      ----- Interrupt -----
0      1001 add $1, %ax      1001 add $1, %ax
0      ----- Interrupt -----      ----- Interrupt -----
1      1002 mov %ax, 2000      1002 mov %ax, 2000
1      ----- Interrupt -----      ----- Interrupt -----
1      1002 mov %ax, 2000      1002 mov %ax, 2000
1      ----- Interrupt -----      ----- Interrupt -----
1      1003 sub $1, %bx      1003 sub $1, %bx
1      ----- Interrupt -----      ----- Interrupt -----
1      1003 sub $1, %bx      1003 sub $1, %bx
1      ----- Interrupt -----      ----- Interrupt -----
1      1004 test $0, %bx      1004 test $0, %bx
1      ----- Interrupt -----      ----- Interrupt -----
1      1004 test $0, %bx      1004 test $0, %bx
1      ----- Interrupt -----      ----- Interrupt -----
1      1005 jgt .top          1005 jgt .top
1      ----- Interrupt -----      ----- Interrupt -----
1      1005 jgt .top          1005 jgt .top
1      ----- Interrupt -----      ----- Interrupt -----
1      1006 halt              1006 halt
1      ----- Halt;Switch -----      ----- Halt;Switch -----
1      1006 halt              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      1000 test $1, %ax      1000 test $1, %ax
0      1      1001 je .signaller      1001 je .signaller
0      1      1006 mov $1, 2000          1002 mov 2000, %cx
1      1      1007 halt              1003 test $1, %cx
1      0      ----- Halt;Switch -----      ----- Halt;Switch -----
1      0      1000 test $1, %ax      1004 jne .waiter
1      0      1001 je .signaller      1005 halt
1      0      1002 mov 2000, %cx      1006 mov $1, %cx
1      0      1003 test $1, %cx      1007 halt
1      0      1004 jne .waiter      1008 halt
1      0      1005 halt              1009 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      1000 test $1, %ax      1000 test $1, %ax
0      0      1001 je .signaller      1001 je .signaller
0      0      1002 mov 2000, %cx      1002 mov 2000, %cx
0      0      1003 test $1, %cx      1003 test $1, %cx
0      0      1004 jne .waiter      1004 jne .waiter
0      0      1002 mov 2000, %cx      1005 jne .waiter
0      0      1003 test $1, %cx      1006 mov 2000, %cx
0      0      1004 jne .waiter      1007 test $1, %cx
0      0      1002 mov 2000, %cx      1008 jne .waiter
0      0      1003 test $1, %cx      1009 mov 2000, %cx
0      0      1004 jne .waiter      1010 test $1, %cx
0      0      1002 mov 2000, %cx      1011 jne .waiter
0      0      1003 test $1, %cx      1012 mov 2000, %cx
0      0      1004 jne .waiter      1013 test $1, %cx
0      0      1002 mov 2000, %cx      1014 jne .waiter
0      0      1003 test $1, %cx      1015 mov 2000, %cx
0      0      1004 jne .waiter      1016 test $1, %cx
0      0      1002 mov 2000, %cx      1017 jne .waiter
0      0      1003 test $1, %cx      1018 mov 2000, %cx
0      0      1004 jne .waiter      1019 test $1, %cx
0      0      1002 mov 2000, %cx      1020 jne .waiter
0      0      1003 test $1, %cx      1021 mov 2000, %cx
0      0      1004 jne .waiter      1022 test $1, %cx
0      0      1002 mov 2000, %cx      1023 jne .waiter
0      0      1003 test $1, %cx      1024 mov 2000, %cx
0      0      1004 jne .waiter      1025 test $1, %cx
0      0      1002 mov 2000, %cx      1026 jne .waiter
0      0      1003 test $1, %cx      1027 mov 2000, %cx
0      0      1004 jne .waiter      1028 test $1, %cx
0      0      1002 mov 2000, %cx      1029 jne .waiter
0      0      1003 test $1, %cx      1030 mov 2000, %cx
0      0      1004 jne .waiter      1031 test $1, %cx
0      0      1002 mov 2000, %cx      1032 jne .waiter
0      0      1003 test $1, %cx      1033 mov 2000, %cx
0      0      1004 jne .waiter      1034 test $1, %cx
0      0      1002 mov 2000, %cx      1035 jne .waiter
0      0      1003 test $1, %cx      1036 mov 2000, %cx
0      0      1004 jne .waiter      1037 test $1, %cx
0      0      1002 mov 2000, %cx      1038 jne .waiter
0      0      1003 test $1, %cx      1039 mov 2000, %cx
0      0      1004 jne .waiter      1040 test $1, %cx
0      0      1002 mov 2000, %cx      1041 jne .waiter
0      0      1003 test $1, %cx      1042 mov 2000, %cx
0      0      1004 jne .waiter      1043 test $1, %cx
0      0      1002 mov 2000, %cx      1044 jne .waiter
0      0      1003 test $1, %cx      1045 mov 2000, %cx
0      0      1004 jne .waiter      1046 test $1, %cx
0      0      1002 mov 2000, %cx      1047 jne .waiter
0      0      1003 test $1, %cx      1048 mov 2000, %cx
0      0      1004 jne .waiter      1049 test $1, %cx
0      0      1002 mov 2000, %cx      1050 jne .waiter
0      0      1003 test $1, %cx      1051 mov 2000, %cx
0      0      1004 jne .waiter      1052 test $1, %cx
0      0      1002 mov 2000, %cx      1053 jne .waiter
0      0      1003 test $1, %cx      1054 mov 2000, %cx
0      0      1004 jne .waiter      1055 test $1, %cx
0      0      1002 mov 2000, %cx      1056 jne .waiter
0      0      1003 test $1, %cx      1057 mov 2000, %cx
0      0      1004 jne .waiter      1058 test $1, %cx
0      0      1002 mov 2000, %cx      1059 jne .waiter
0      0      1003 test $1, %cx      1060 mov 2000, %cx
0      0      1004 jne .waiter      1061 test $1, %cx
0      0      1002 mov 2000, %cx      1062 jne .waiter
0      0      1003 test $1, %cx      1063 mov 2000, %cx
0      0      1004 jne .waiter      1064 test $1, %cx
0      0      1002 mov 2000, %cx      1065 jne .waiter
0      0      1003 test $1, %cx      1066 mov 2000, %cx
0      0      1004 jne .waiter      1067 test $1, %cx
0      0      1002 mov 2000, %cx      1068 jne .waiter
0      0      1003 test $1, %cx      1069 mov 2000, %cx
0      0      1004 jne .waiter      1070 test $1, %cx
0      0      1002 mov 2000, %cx      1071 jne .waiter
0      0      1003 test $1, %cx      1072 mov 2000, %cx
0      0      1004 jne .waiter      1073 test $1, %cx
0      0      1002 mov 2000, %cx      1074 jne .waiter
0      0      1003 test $1, %cx      1075 mov 2000, %cx
0      0      1004 jne .waiter      1076 test $1, %cx
0      0      1002 mov 2000, %cx      1077 jne .waiter
0      0      1003 test $1, %cx      1078 mov 2000, %cx
0      0      1004 jne .waiter      1079 test $1, %cx
0      0      1002 mov 2000, %cx      1080 jne .waiter
0      0      1003 test $1, %cx      1081 mov 2000, %cx
0      0      1004 jne .waiter      1082 test $1, %cx
0      0      1002 mov 2000, %cx      1083 jne .waiter
0      0      1003 test $1, %cx      1084 mov 2000, %cx
0      0      1004 jne .waiter      1085 test $1, %cx
0      0      1002 mov 2000, %cx      1086 jne .waiter
0      0      1003 test $1, %cx      1087 mov 2000, %cx
0      0      1004 jne .waiter      1088 test $1, %cx
0      0      1002 mov 2000, %cx      1089 jne .waiter
0      0      1003 test $1, %cx      1090 mov 2000, %cx
0      0      1004 jne .waiter      1091 test $1, %cx
0      0      1002 mov 2000, %cx      1092 jne .waiter
0      0      1003 test $1, %cx      1093 mov 2000, %cx
0      0      1004 jne .waiter      1094 test $1, %cx
0      0      1002 mov 2000, %cx      1095 jne .waiter
0      0      1003 test $1, %cx      1096 mov 2000, %cx
0      0      1004 jne .waiter      1097 test $1, %cx
0      0      1002 mov 2000, %cx      1098 jne .waiter
0      0      1003 test $1, %cx      1099 mov 2000, %cx
0      0      1004 jne .waiter      1100 test $1, %cx
0      0      1002 mov 2000, %cx      1101 jne .waiter
0      0      1003 test $1, %cx      1102 mov 2000, %cx
0      0      1004 jne .waiter      1103 test $1, %cx
0      0      1002 mov 2000, %cx      1104 jne .waiter
0      0      1003 test $1, %cx      1105 mov 2000, %cx
0      0      1004 jne .waiter      1106 test $1, %cx
0      0      1002 mov 2000, %cx      1107 jne .waiter
0      0      1003 test $1, %cx      1108 mov 2000, %cx
0      0      1004 jne .waiter      1109 test $1, %cx
0      0      1002 mov 2000, %cx      1110 jne .waiter
0      0      1003 test $1, %cx      1111 mov 2000, %cx
0      0      1004 jne .waiter      1112 test $1, %cx
0      0      1002 mov 2000, %cx      1113 jne .waiter
0      0      1003 test $1, %cx      1114 mov 2000, %cx
0      0      1004 jne .waiter      1115 test $1, %cx
0      0      1002 mov 2000, %cx      1116 jne .waiter
0      0      1003 test $1, %cx      1117 mov 2000, %cx
0      0      1004 jne .waiter      1118 test $1, %cx
0      0      1002 mov 2000, %cx      1119 jne .waiter
0      0      1003 test $1, %cx      1120 mov 2000, %cx
0      0      1004 jne .waiter      1121 test $1, %cx
0      0      1002 mov 2000, %cx      1122 jne .waiter
0      0      1003 test $1, %cx      1123 mov 2000, %cx
0      0      1004 jne .waiter      1124 test $1, %cx
0      0      1002 mov 2000, %cx      1125 jne .waiter
0      0      1003 test $1, %cx      1126 mov 2000, %cx
0      0      1004 jne .waiter      1127 test $1, %cx
0      0      1002 mov 2000, %cx      1128 jne .waiter
0      0      1003 test $1, %cx      1129 mov 2000, %cx
0      0      1004 jne .waiter      1130 test $1, %cx
0      0      1002 mov 2000, %cx      1131 jne .waiter
0      0      1003 test $1, %cx      1132 mov 2000, %cx
0      0      1004 jne .waiter      1133 test $1, %cx
0      0      1002 mov 2000, %cx      1134 jne .waiter
0      0      1003 test $1, %cx      1135 mov 2000, %cx
0      0      1004 jne .waiter      1136 test $1, %cx
0      0      1002 mov 2000, %cx      1137 jne .waiter
0      0      1003 test $1, %cx      1138 mov 2000, %cx
0      0      1004 jne .waiter      1139 test $1, %cx
0      0      1002 mov 2000, %cx      1140 jne .waiter
0      0      1003 test $1, %cx      1141 mov 2000, %cx
0      0      1004 jne .waiter      1142 test $1, %cx
0      0      1002 mov 2000, %cx      1143 jne .waiter
0      0      1003 test $1, %cx      1144 mov 2000, %cx
0      0      1004 jne .waiter      1145 test $1, %cx
0      0      1002 mov 2000, %cx      1146 jne .waiter
0      0      1003 test $1, %cx      1147 mov 2000, %cx
0      0      1004 jne .waiter      1148 test $1, %cx
0      0      1002 mov 2000, %cx      1149 jne .waiter
0      0      1003 test $1, %cx      1150 mov 2000, %cx
0      0      1004 jne .waiter      1151 test $1, %cx
0      0      1002 mov 2000, %cx      1152 jne .waiter
0      0      1003 test $1, %cx      1153 mov 2000, %cx
0      0      1004 jne .waiter      1154 test $1, %cx
0      0      1002 mov 2000, %cx      1155 jne .waiter
0      0      1003 test $1, %cx      1156 mov 2000, %cx
0      0      1004 jne .waiter      1157 test $1, %cx
0      0      1002 mov 2000, %cx      1158 jne .waiter
0      0      1003 test $1, %cx      1159 mov 2000, %cx
0      0      1004 jne .waiter      1160 test $1, %cx
0      0      1002 mov 2000, %cx      1161 jne .waiter
0      0      1003 test $1, %cx      1162 mov 2000, %cx
0      0      1004 jne .waiter      1163 test $1, %cx
0      0      1002 mov 2000, %cx      1164 jne .waiter
0      0      1003 test $1, %cx      1165 mov 2000, %cx
0      0      1004 jne .waiter      1166 test $1, %cx
0      0      1002 mov 2000, %cx      1167 jne .waiter
0      0      1003 test $1, %cx      1168 mov 2000, %cx
0      0      1004 jne .waiter      1169 test $1, %cx
0      0      1002 mov 2000, %cx      1170 jne .waiter
0      0      1003 test $1, %cx      1171 mov 2000, %cx
0      0      1004 jne .waiter      1172 test $1, %cx
0      0      1002 mov 2000, %cx      1173 jne .waiter
0      0      1003 test $1, %cx      1174 mov 2000, %cx
0      0      1004 jne .waiter      1175 test $1, %cx
0      0      1002 mov 2000, %cx      1176 jne .waiter
0      0      1003 test $1, %cx      1177 mov 2000, %cx
0      0      1004 jne .waiter      1178 test $1, %cx
0      0      1002 mov 2000, %cx      1179 jne .waiter
0      0      1003 test $1, %cx      1180 mov 2000, %cx
0      0      1004 jne .waiter      1181 test $1, %cx
0      0      1002 mov 2000, %cx      1182 jne .waiter
0      0      1003 test $1, %cx      1183 mov 2000, %cx
0      0      1004 jne .waiter      1184 test $1, %cx
```