

## 1. 复现文档中的例子

simple-race.s

```
ocelot@ubuntu: ~/HW-ThreadsIntro
1000 mov 2000(%bx), %ax
1001 add $1, %ax
1002 mov %ax, 2000(%bx)
1003 halt
ocelot@ubuntu:~/HW-ThreadsIntro$ python --v
Command 'python' not found, did you mean:
  command 'python3' from deb python3
  command 'python' from deb python-is-python3
ocelot@ubuntu:~/HW-ThreadsIntro$ python-is-python3
python-is-python3: command not found
ocelot@ubuntu:~/HW-ThreadsIntro$ python2 x86.py -p simple-race.s -t 1 -M 2000 -R ax,bx -c
ARG seed 0
ARG numthreads 1
ARG program simple-race.s
ARG interrupt frequency 50
ARG interrupt randomness False
ARG argv
ARG load address 1000
ARG memsize 128
ARG memtrace 2000
ARG regtrace ax,bx
ARG cctrace False
ARG printstats False
ARG verbose False

2000      ax      bx      Thread 0
0         0       0
0         0       0  1000 mov 2000(%bx), %ax
0         1       0  1001 add $1, %ax
1         1       0  1002 mov %ax, 2000(%bx)
1         1       0  1003 halt
ocelot@ubuntu:~/HW-ThreadsIntro$
```

如图，就算结果正确。

loop.s

```
ocelot@ubuntu: ~/HW-ThreadsIntro
1      1      0  1003 halt
ocelot@ubuntu:~/HW-ThreadsIntro$ python2 x86.py -p loop.s -t 1 -a dx=3 -R dx -C -c
ARG seed 0
ARG numthreads 1
ARG program loop.s
ARG interrupt frequency 50
ARG interrupt randomness False
ARG argv dx=3
ARG load address 1000
ARG memsize 128
ARG memtrace
ARG regtrace dx
ARG cctrace True
ARG printstats False
ARG verbose False

dx  >= > <= < != ==      Thread 0
3  0  0  0  0  0  0  0
2  0  0  0  0  0  0  0  1000 sub $1,%dx
2  1  1  0  0  1  0  0  1001 test $0,%dx
2  1  1  0  0  1  0  0  1002 jgte .top
1  1  1  0  0  1  0  0  1000 sub $1,%dx
1  1  1  0  0  1  0  0  1001 test $0,%dx
1  1  1  0  0  1  0  0  1002 jgte .top
0  1  1  0  0  1  0  0  1000 sub $1,%dx
0  1  0  1  0  0  1  0  1001 test $0,%dx
0  1  0  1  0  0  1  0  1002 jgte .top
-1 1  0  1  0  0  1  0  1000 sub $1,%dx
-1 0  0  1  1  1  0  0  1001 test $0,%dx
-1 0  0  1  1  1  0  0  1002 jgte .top
-1 0  0  1  1  1  0  0  1003 halt
ocelot@ubuntu:~/HW-ThreadsIntro$
```

如图，计算结果正确。

## looping-race-nolock.s

```
ocelot@ubuntu: ~/HW-ThreadsIntro
ARG seed 0
ARG numthreads 2
ARG program looping-race-nolock.s
ARG interrupt frequency 50
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
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
1 ----- Halt;Switch ----- ----- Halt;Switch -----
1 1000 mov 2000, %ax
1 1001 add $1, %ax
2 1002 mov %ax, 2000
2 1003 sub $1, %bx
2 1004 test $0, %bx
2 1005 jgt .top
2 1006 halt
ocelot@ubuntu: ~/HW-ThreadsIntro$
```

如图，不使用中断时，计算结果为 2。

```
ocelot@ubuntu: ~/HW-ThreadsIntro
ARG load address 1000
ARG memsize 128
ARG memtrace 2000
ARG regtrace ax,bx
ARG cctrace False
ARG printstats False
ARG verbose False

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

如图所示，使用中断时，出现了竞争，计算结果仅为 1。

## 2. 编写自己的汇编程序，截屏展示结果

```
# assumes %bx has loop count in it

.main

.top

# critical section

mov 2000, %ax # get 'value' at address 2000

add $1, %ax    # increment it

add $1, %ax

mov %ax, 2000 # store it back


# see if we're still looping

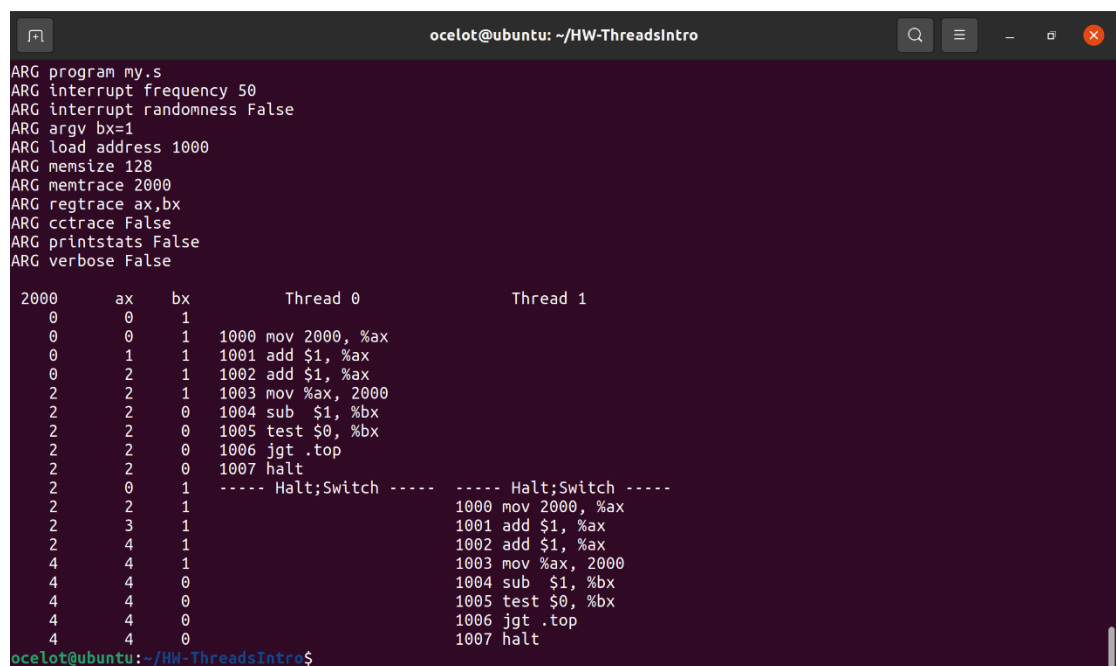
sub  $1, %bx

test $0, %bx

jgt .top


halt
```

在 looping-race-nolock.s 中加一行代码 `add $1, %ax`，分别使用与不使用中断，观察效果：



```
ocelot@ubuntu: ~/HW-ThreadsIntro
ARG program my.s
ARG interrupt frequency 50
ARG interrupt randomness False
ARG argv bx=1
ARG load address 1000
ARG memsize 128
ARG memtrace 2000
ARG regtrace ax,bx
ARG cctrace False
ARG printstats False
ARG verbose False

2000    ax    bx          Thread 0          Thread 1
0       0     1
0       0     1    1000 mov 2000, %ax
0       1     1    1001 add $1, %ax
0       2     1    1002 add $1, %ax
2       2     1    1003 mov %ax, 2000
2       2     0    1004 sub $1, %bx
2       2     0    1005 test $0, %bx
2       2     0    1006 jgt .top
2       2     0    1007 halt
2       0     1    ----- Halt;Switch -----
2       2     1    1000 mov 2000, %ax
2       3     1    1001 add $1, %ax
2       4     1    1002 add $1, %ax
4       4     1    1003 mov %ax, 2000
4       4     0    1004 sub $1, %bx
4       4     0    1005 test $0, %bx
4       4     0    1006 jgt .top
4       4     0    1007 halt
ocelot@ubuntu: ~/HW-ThreadsIntro$
```

```
ocelot@ubuntu: ~/HW-ThreadsIntro
ARG load address 1000
ARG memsize 128
ARG memtrace 2000
ARG regtrace ax,bx
ARG cctrace False
ARG printstats False
ARG verbose False

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

结果与原来的代码类似，有中断时结果为 2，无中断时结果为 4，出现了竞争。