108021209 李思諭

2. Fairness

How “fair” is your scheduler based on the output that you observe?

I think that my scheduler based on the output is very fair. Since my output is A0B1C2D3…, the first output is from Producer1, and the second output is from Producer2, and the third output is from Producer1, and the 4th output is from Producer2, and repeat.

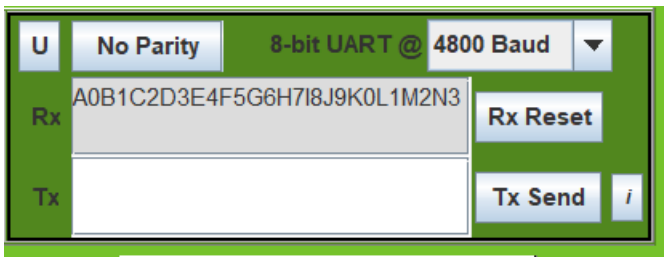
What kinds of output pattern do you consider to be fair?

I think the output pattern A0B1C2D3… is fair.

Does your code have the problem of starvation?

No, my code does not have the problem of starvation.

Propose your own solution to make it fair for these threads. Explain the changes that you made.



I add a variable “turn” in my code.

I initiate “turn” to be 1.

When “turn” is 1, I let Producer1 can add one item into buffer and also make “turn” become 0.

When “turn” is 0, I let Producer2 can add one item into buffer and also make “turn” become 1.

Also, the action of the change of “turn” is in the critical section.

3. Typescript and screenshots

3.1 Typescript for compilation

```
User@LAPTOP-59VRNRON /cygdrive/c/Users/User/Desktop/Home
work/Checkpoint4
$ make clean
rm *.hex *.ihx *.lnk *.lst *.map *.mem *.rel *.rst *.sym
rm: cannot remove '*.ihx': No such file or directory
rm: cannot remove '*.lnk': No such file or directory
make: *** [Makefile:25: clean] Error 1

User@LAPTOP-59VRNRON /cygdrive/c/Users/User/Desktop/Home
work/Checkpoint4
$ make
sdcc -c  test3threads.c
test3threads.c:115: warning 158: overflow in implicit co
nstant conversion
sdcc -c  preemptive.c
preemptive.c:258: warning 85: in function ThreadCreate u
nreferenced function argument : 'fp'
sdcc  -o test3threads.hex test3threads.rel preemptive.re
l

User@LAPTOP-59VRNRON /cygdrive/c/Users/User/Desktop/Home
work/Checkpoint4
```

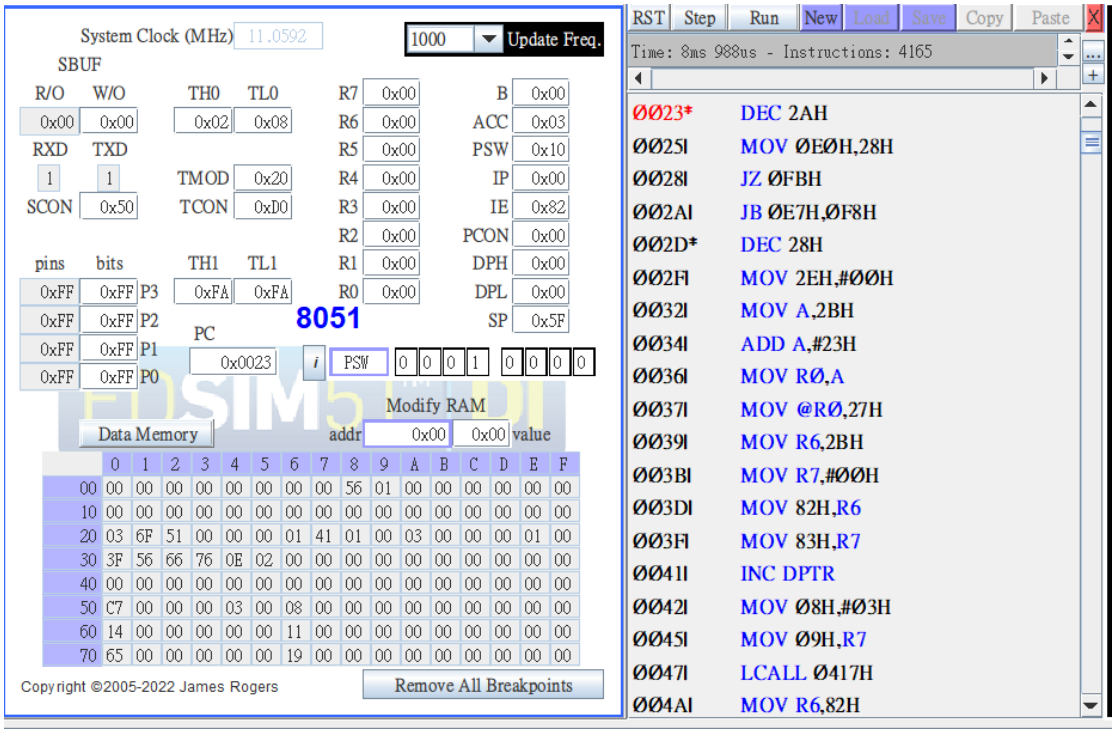
3.2 Screenshots and explanation

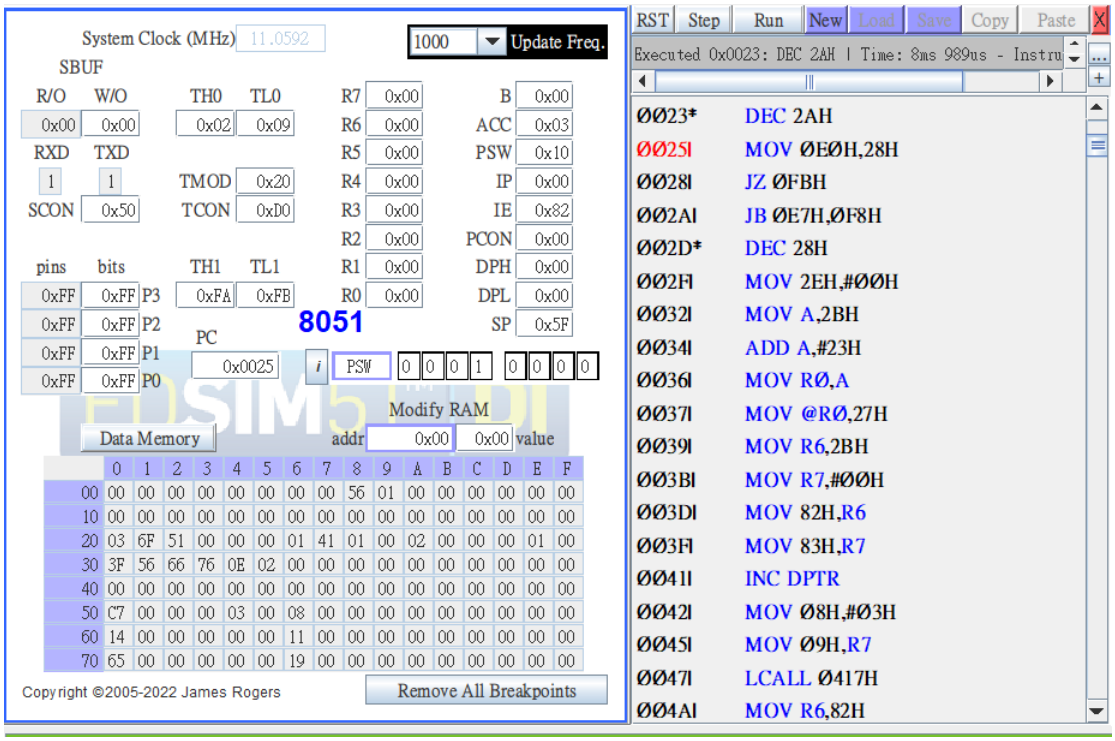
The file “testpreempt.map” shows that the function Producer1 starts from the line 0014, that the function Producer2 starts from the line 0065, and that the function Consumer starts from the line 00B9.

Take screenshots when the Producer1 and Producer2 running and show semaphore changes.

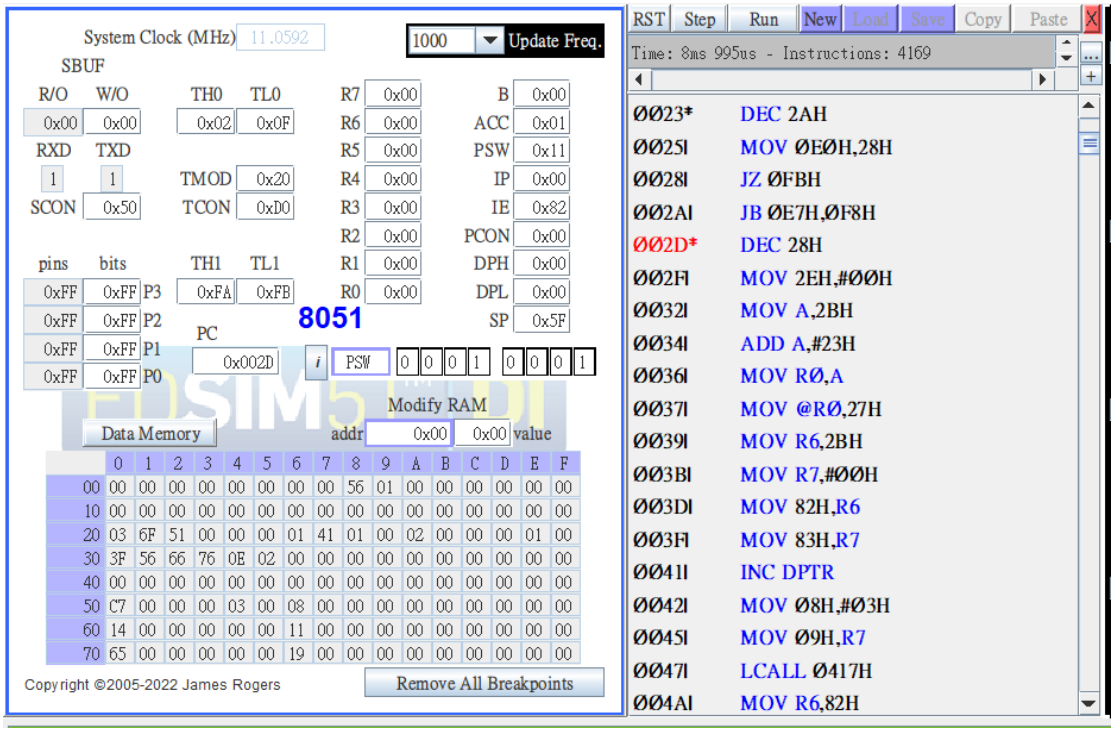
In Producer1 :

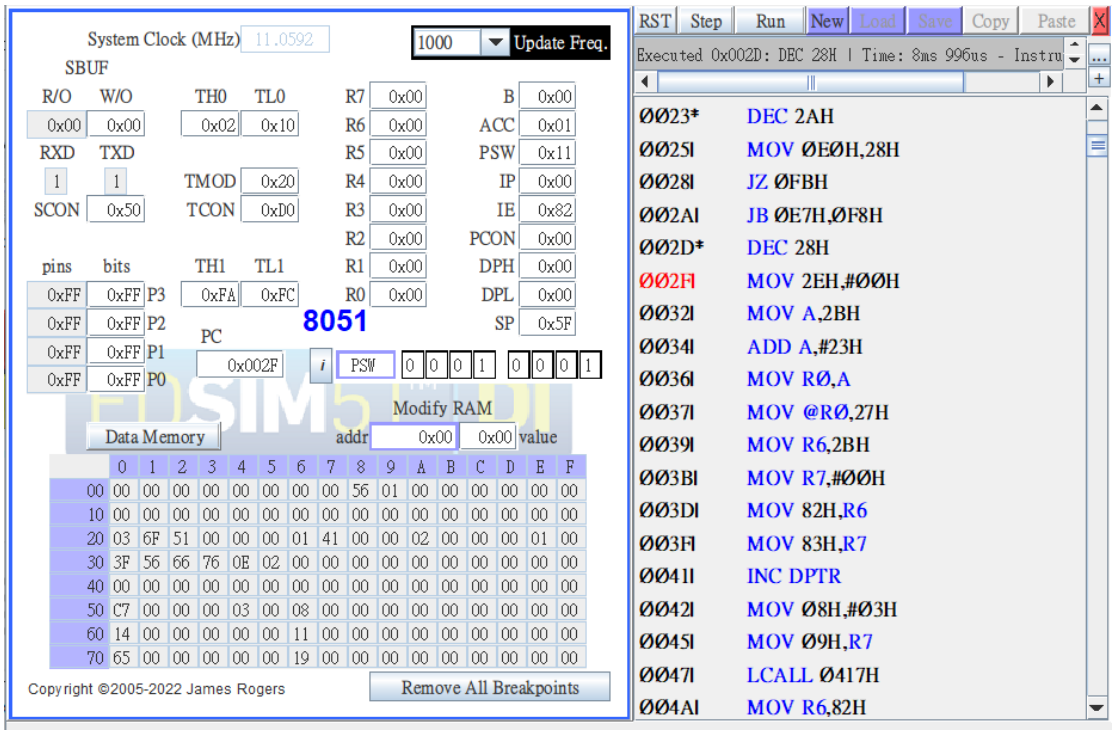
When executing DEC 2AH in the line 0023, it decreases the value of empty by one since it is doing SemaphoreWait(empty).



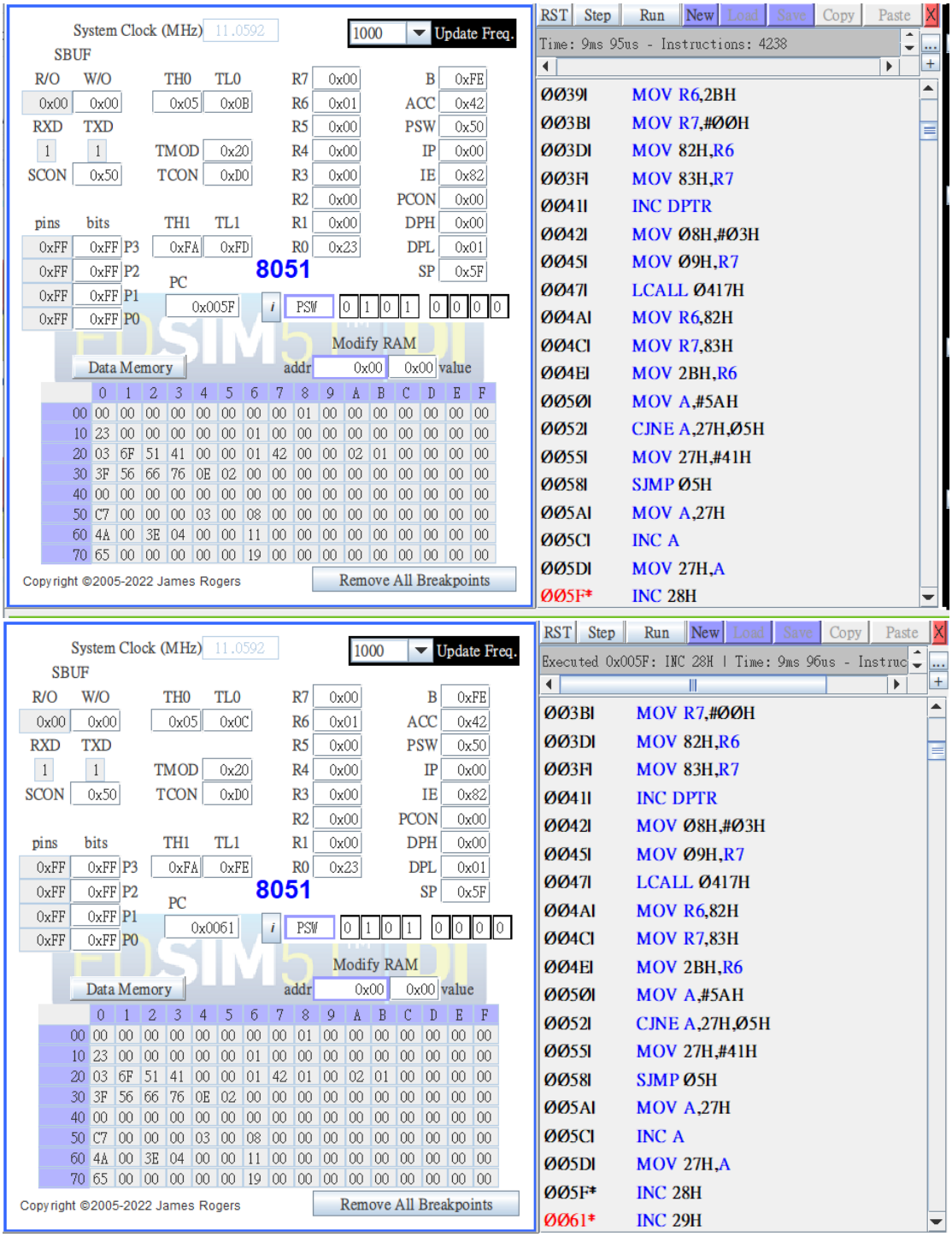


When executing DEC 28H in the line 002D, it decreases the value of mutex by one since it is doing SemaphoreWait(mutex).

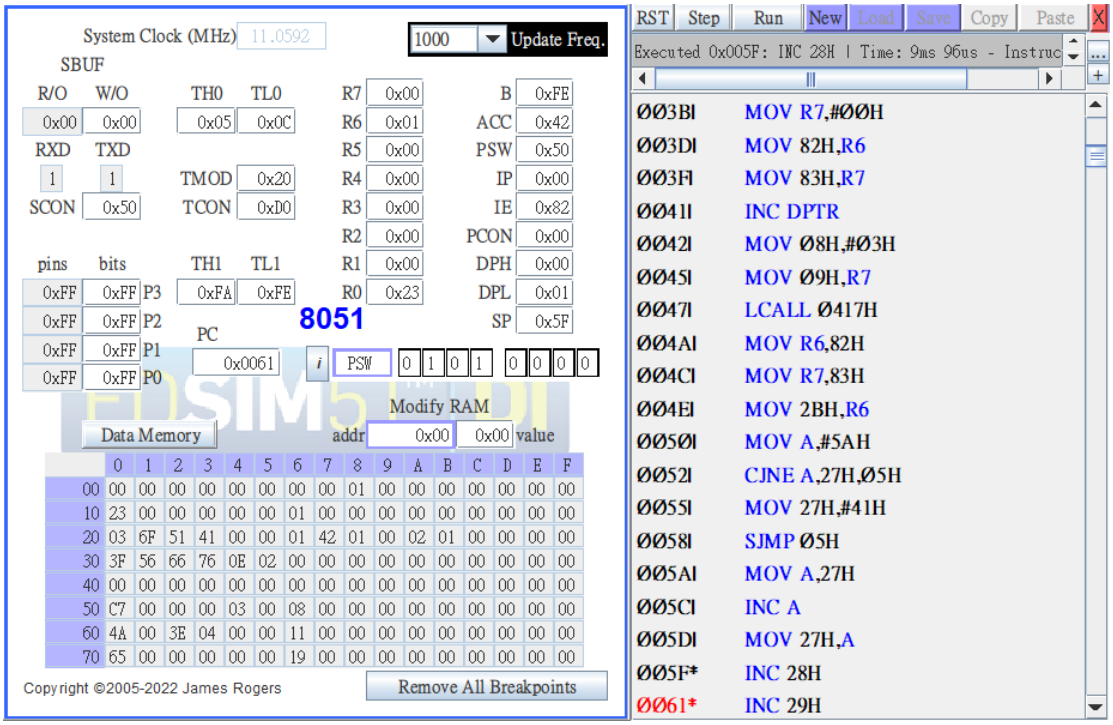


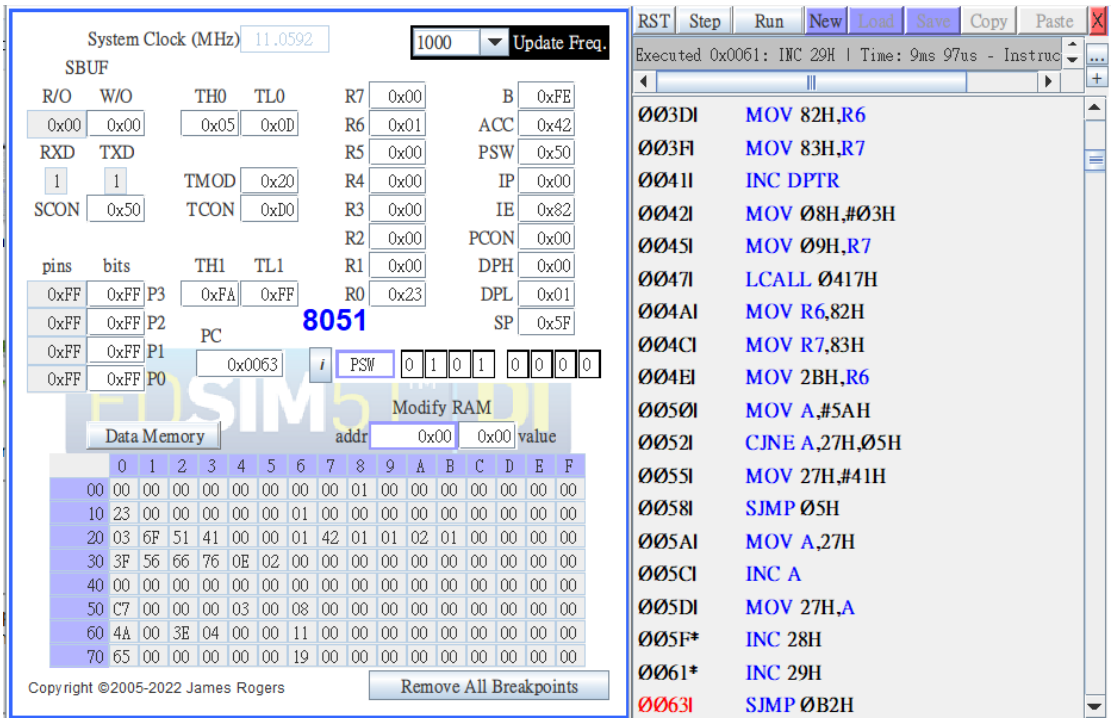


When executing INC 28H in the line 005F, it increases the value of mutex by one since it is doing SemaphoreSignal(mutex).



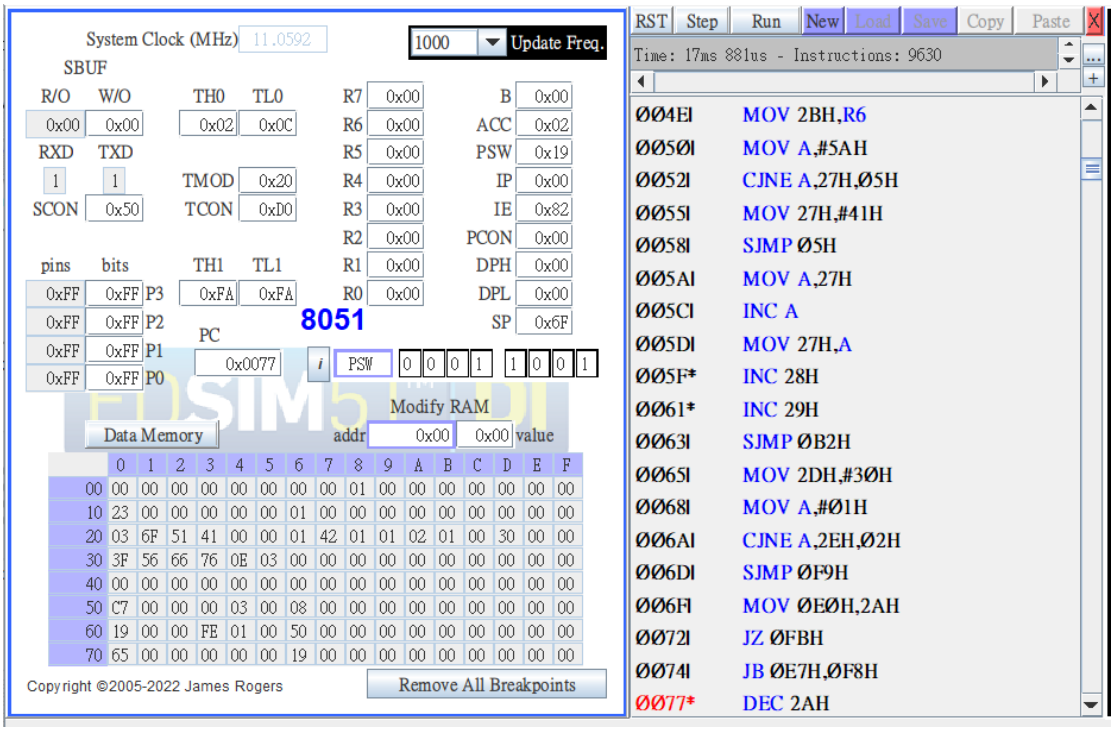
When executing INC 29H in the line 0061, it increases the value of full by one since it is doing SemaphoreSignal(full).

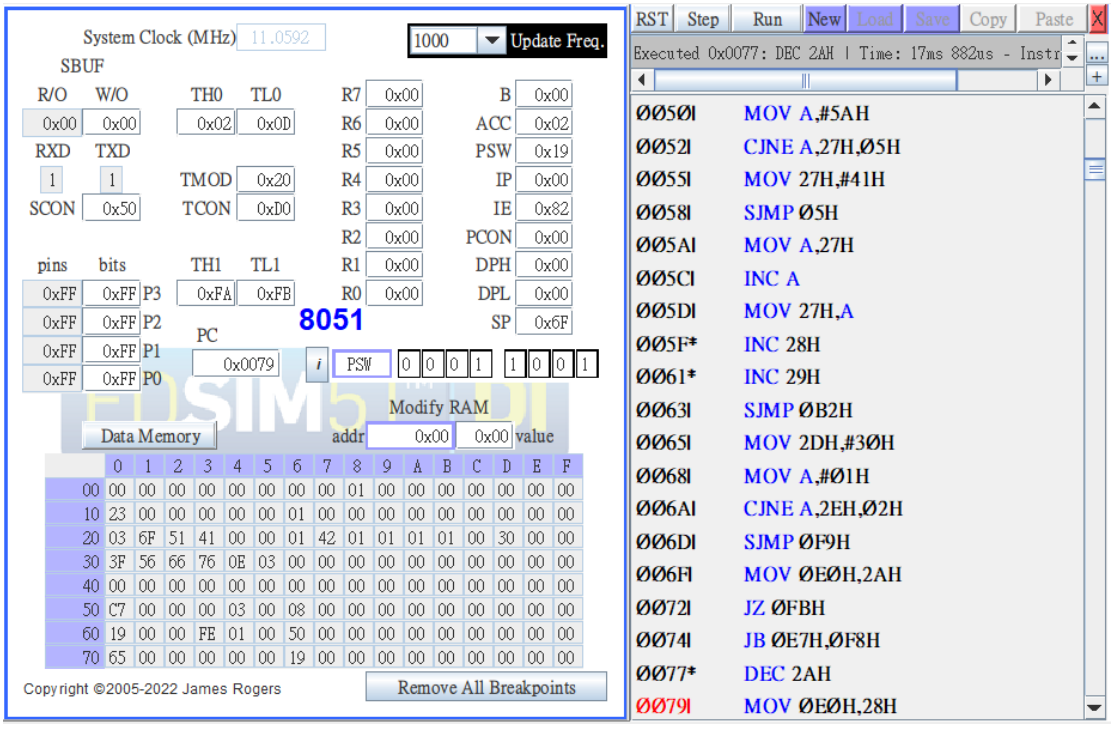




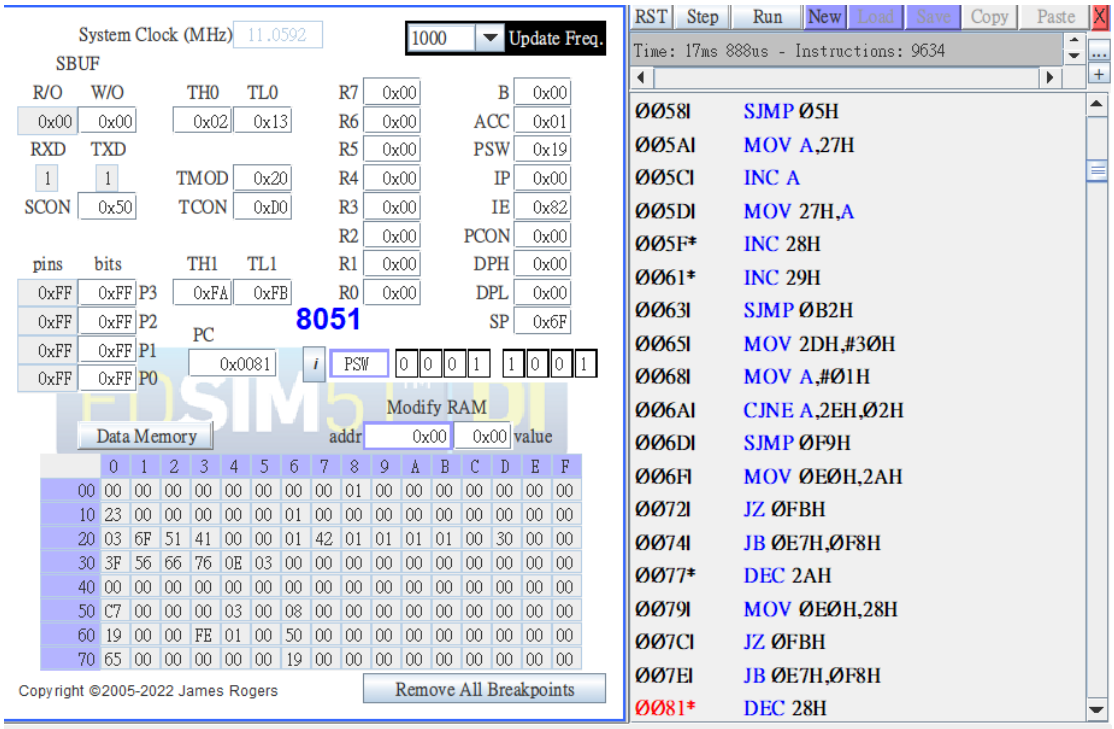
In Producer2 :

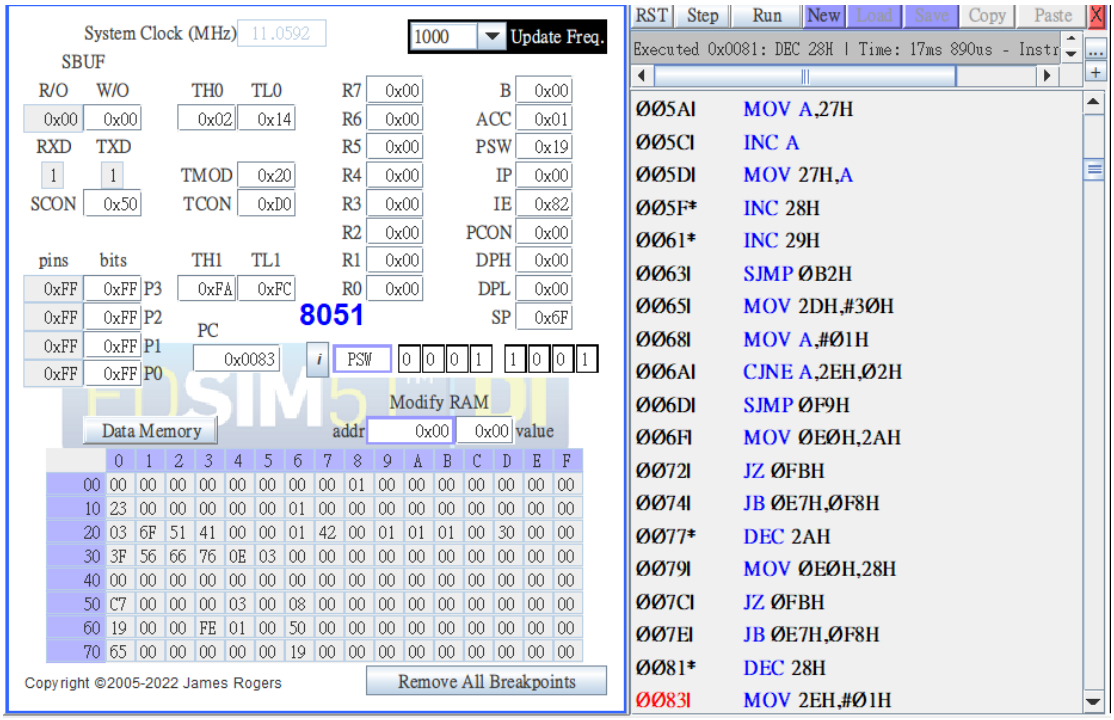
When executing DEC 2AH in the line 0077, it decreases the value of empty by one since it is doing SemaphoreWait(empty).



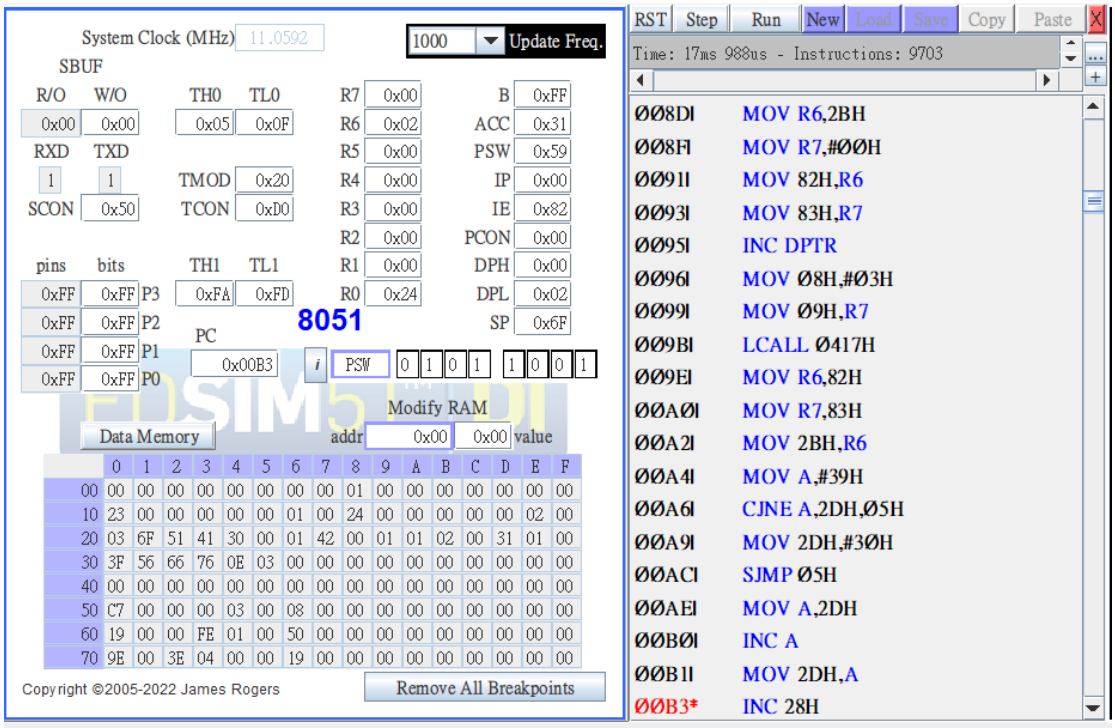


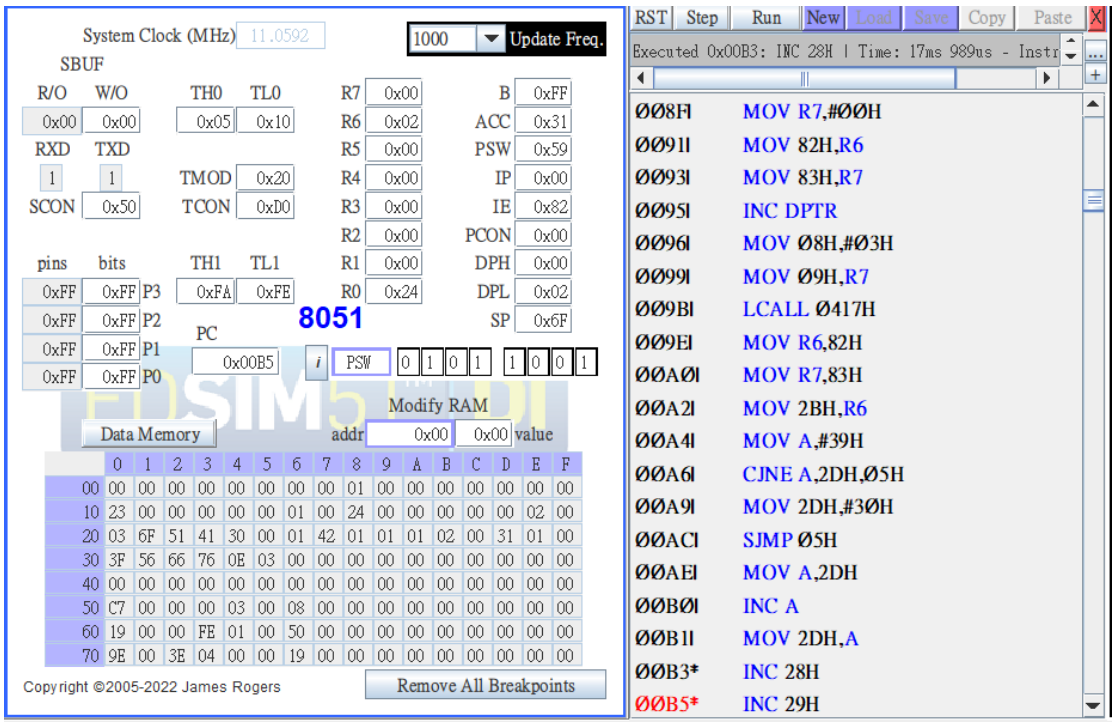
When executing DEC 28H in the line 0081, it decreases the value of mutex by one since it is doing SemaphoreWait(mutex).



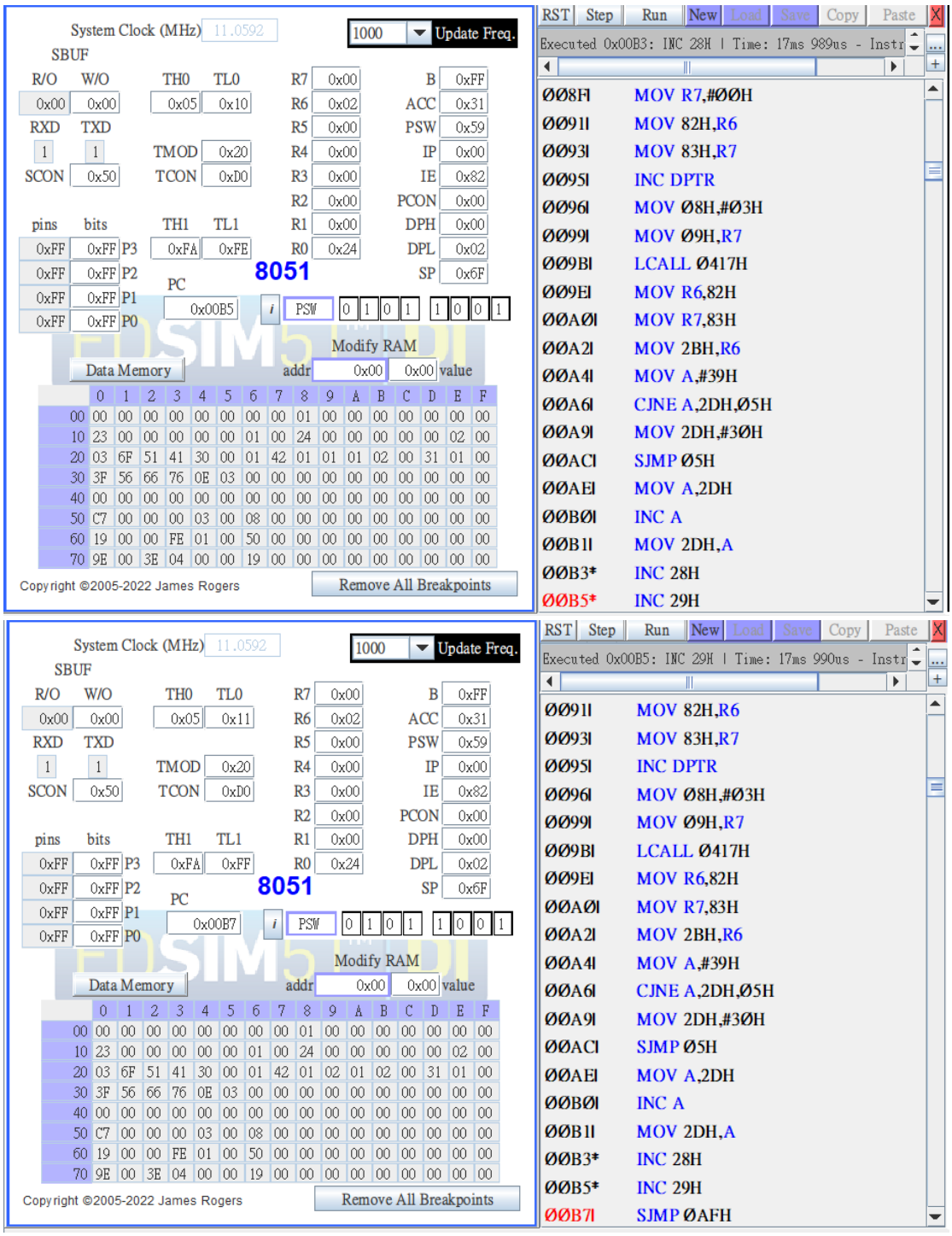


When executing INC 28H in the line 00B3, it increases the value of mutex by one since it is doing SemaphoreSignal(mutex).

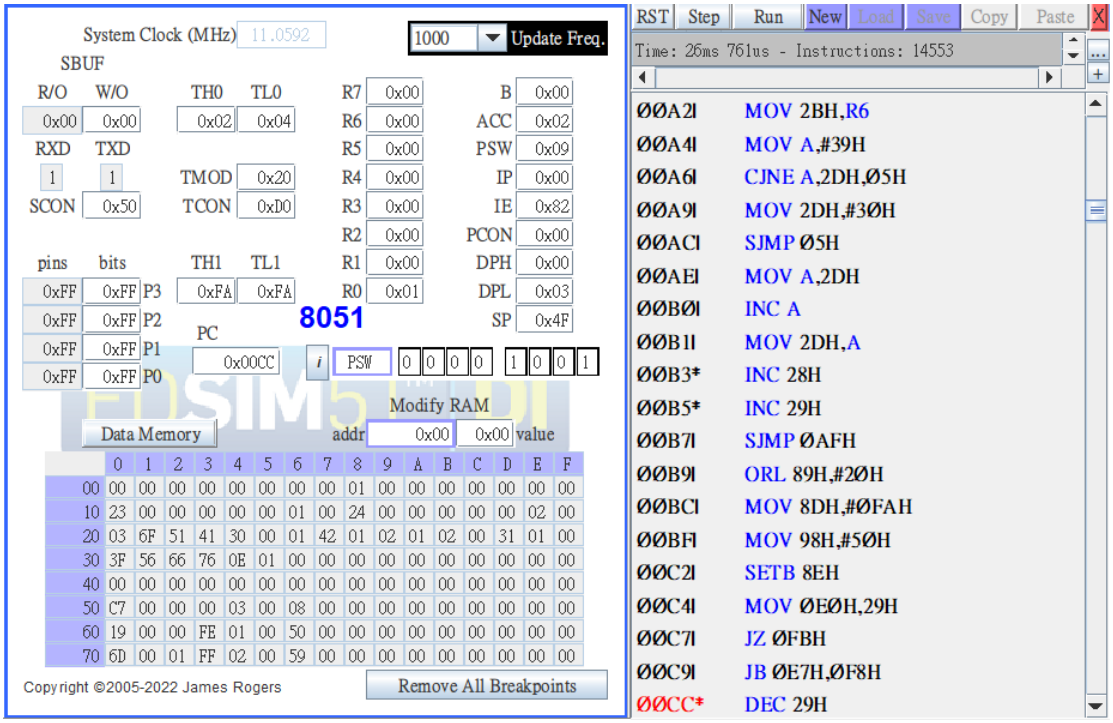


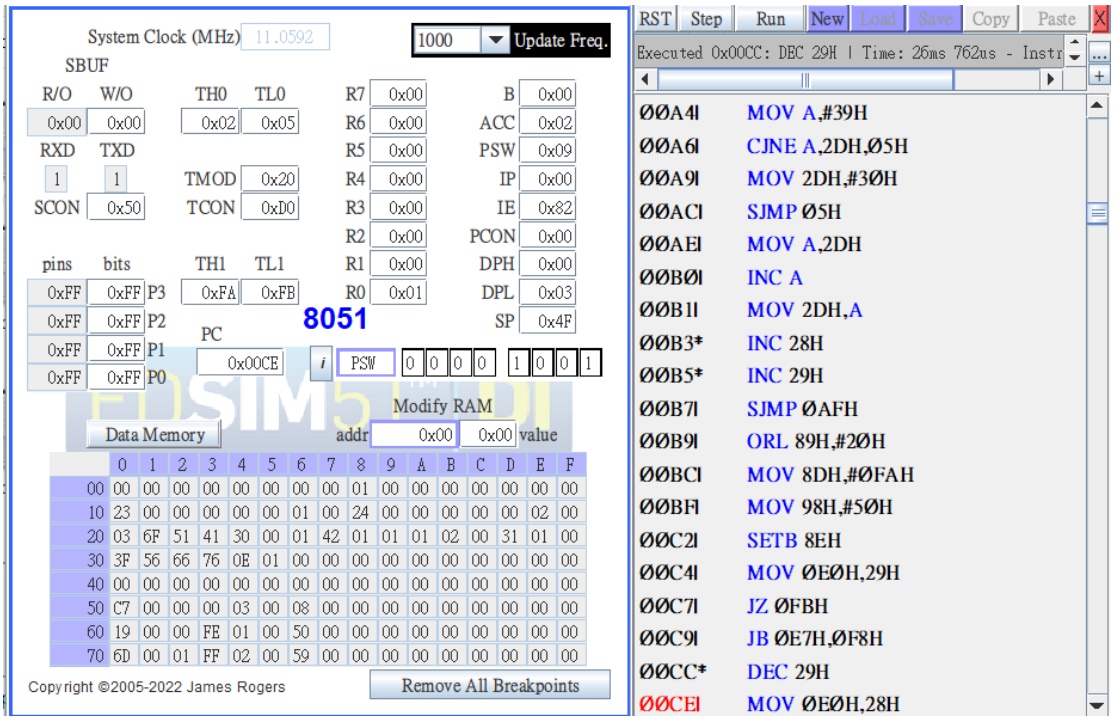


When executing INC 29H in the line 00B5, it increases the value of full by one since it is doing SemaphoreSignal(full).

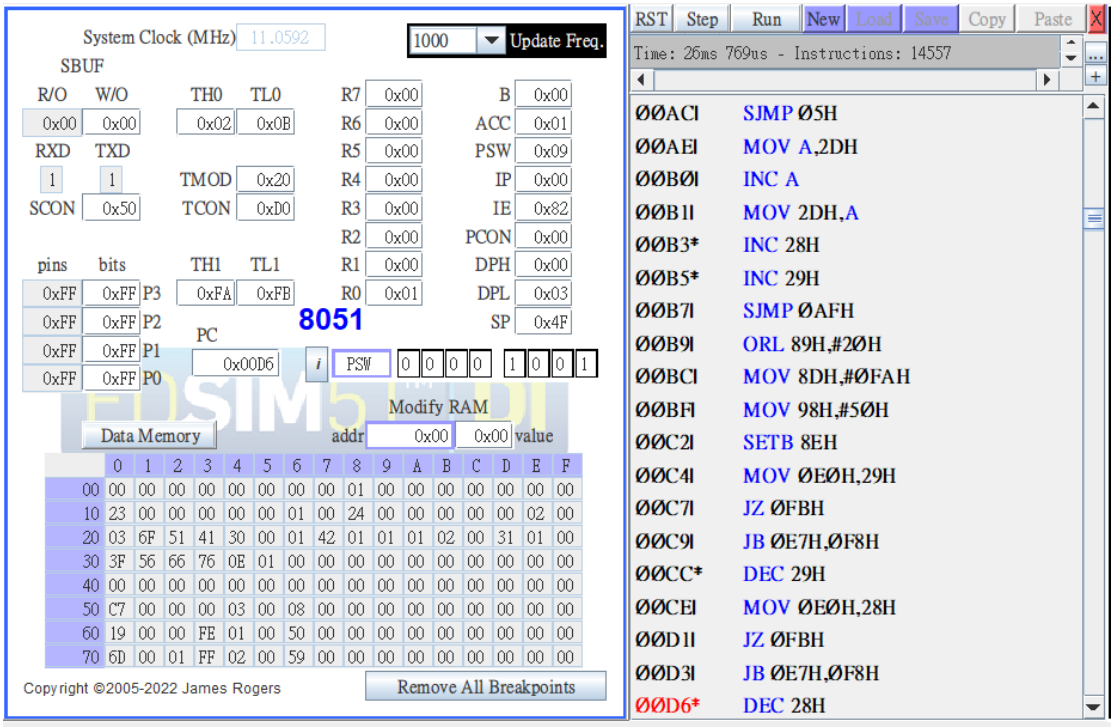


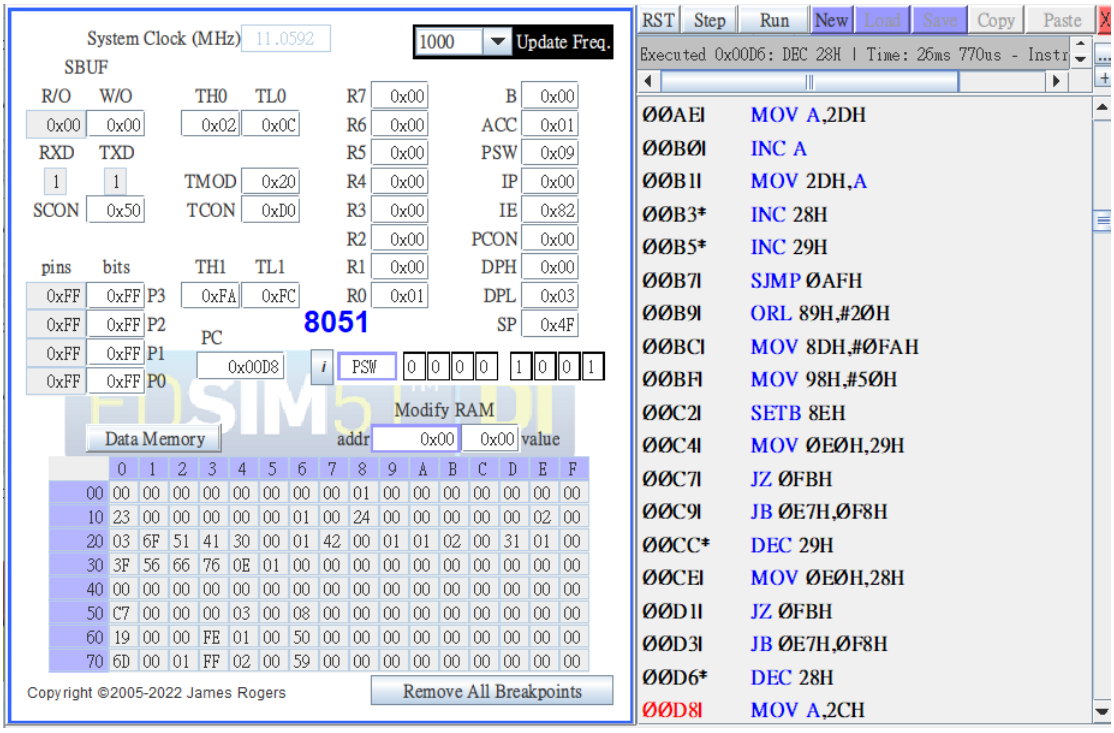
Take screenshots when the Consumer is running and show semaphore changes. When executing DEC 29H in the line 00CC, it decreases the value of full by one since it is doing SemaphoreWait(full).



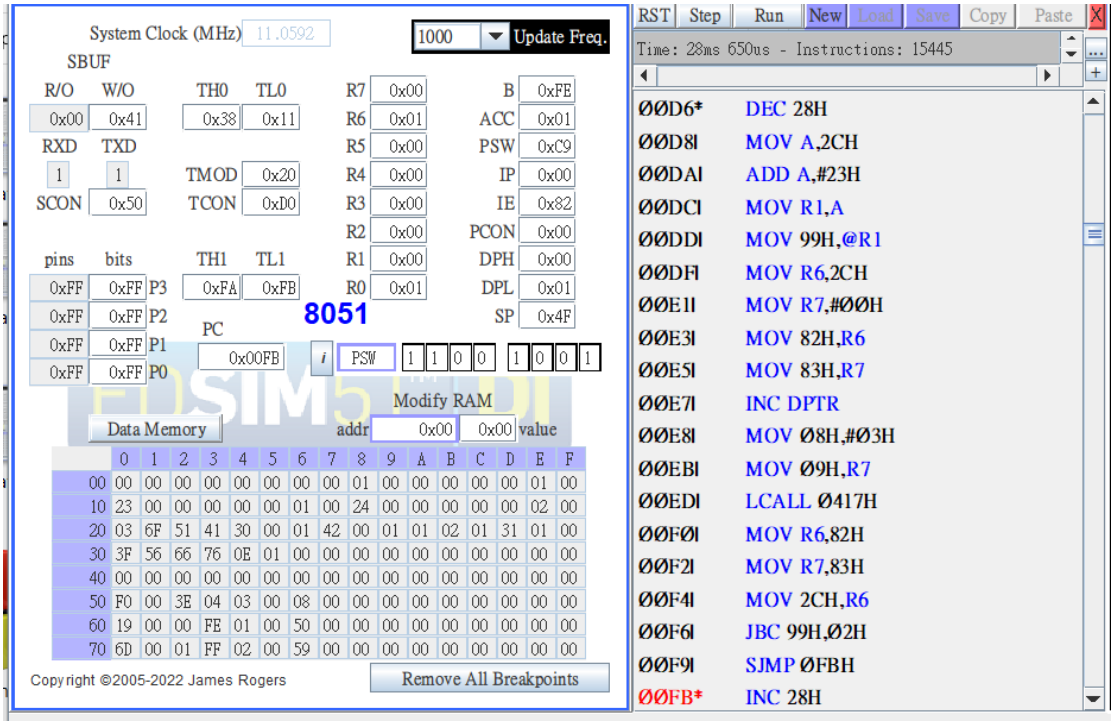


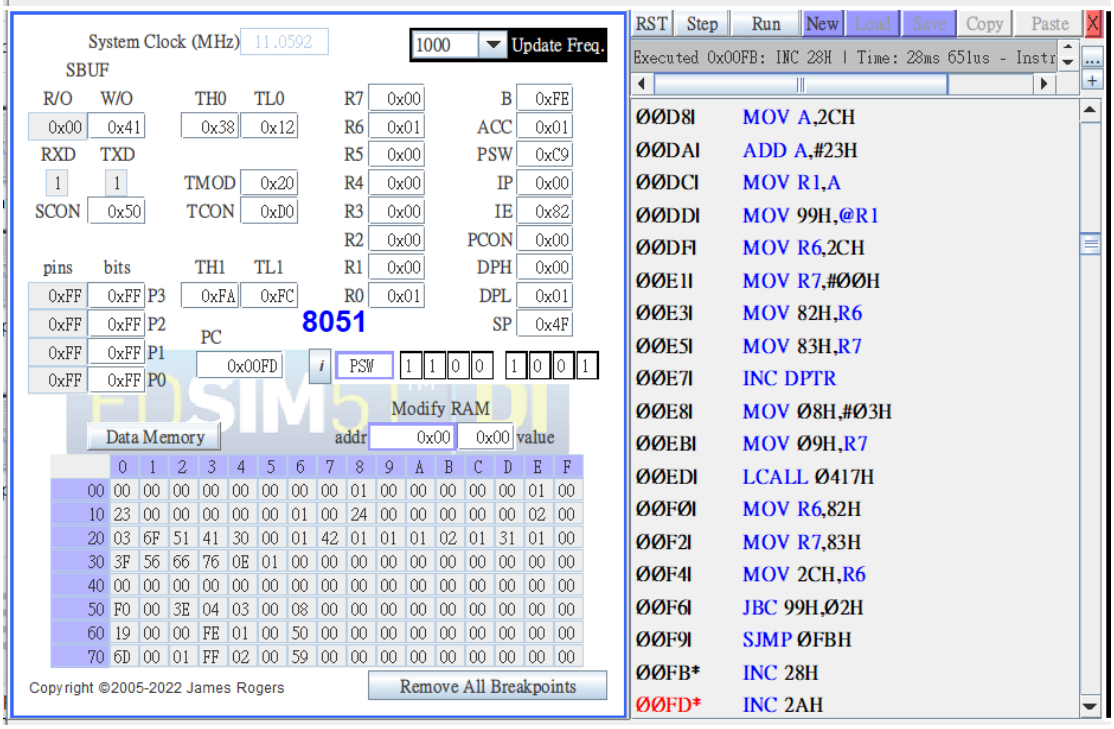
When executing DEC 28H in the line 00D6, it decreases the value of mutex by one since it is doing SemaphoreWait(mutex).



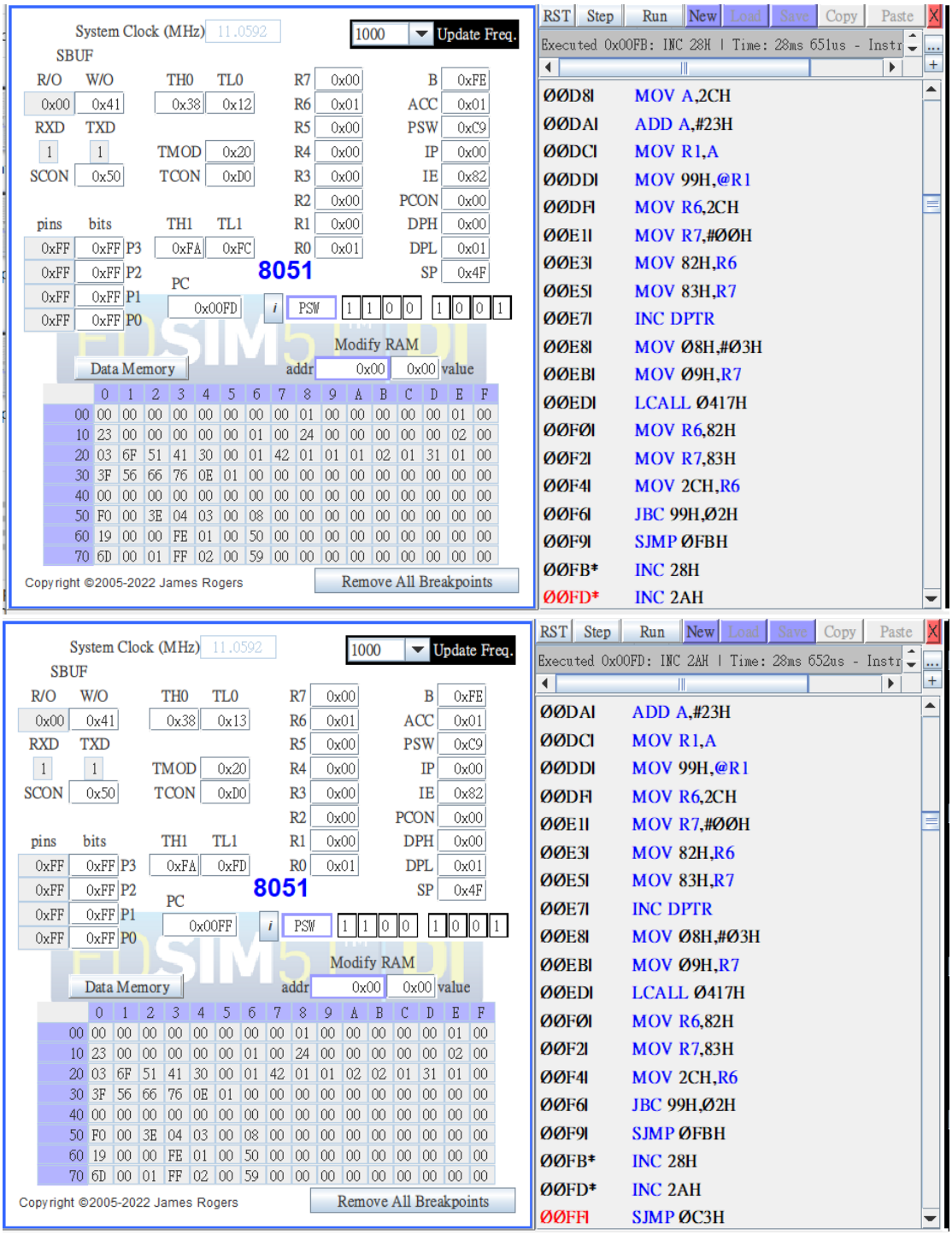


When executing INC 28H in the line 00FB, it increases the value of mutex by one since it is doing SemaphoreSignal(mutex).



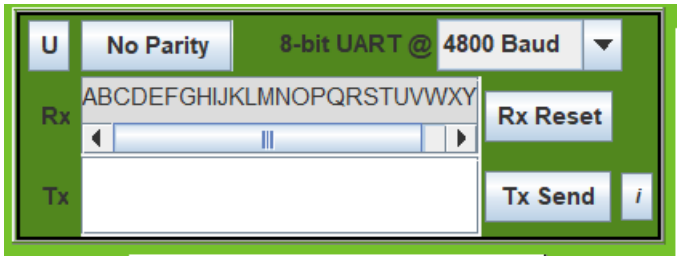


When executing INC 2AH in the line 00FD, it increases the value of empty by one since it is doing SemaphoreSignal(empty).



Show and explain UART output to show the unfair version, if any, and the fair version.

The unfair version :



The reason why causes the unfair reason is that the variable “empty” changes from 3 to 0 in the Producer1, and changes from 0 to 3 in the Consumer. Thus, the Producer2 cannot add its item into buffer since it executes the line “SemaphoreWait(empty);” repeatly.

In the fair version, I solve this problem by giving each Producer one time to add its item to the buffer.

The fair version :