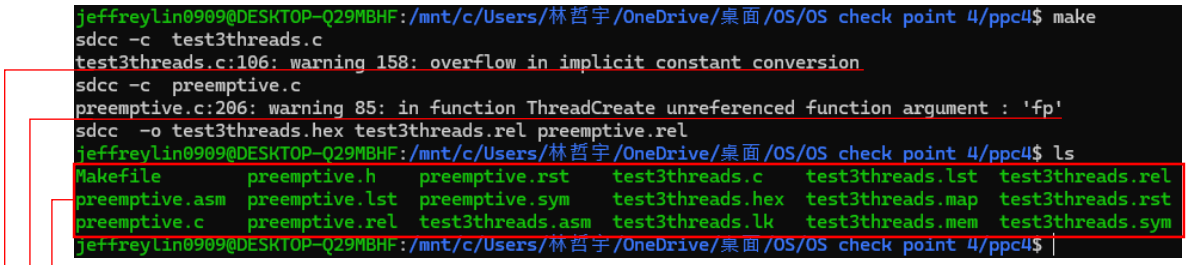
Programming Project Checkpoint 4

111062311 林哲宇

1. Screenshot for compilation:



- The warning message is because of setting TH1 to negative number (-6 for UART baud rate), and it's totally safe.
- The warning message is because of using DPH/DPL instead of identifier of parameter, and it's totally safe.
- Files generated after compilation (including .hex and .map).

Note: for better understanding for following explanation, here's the variable address map of my code:

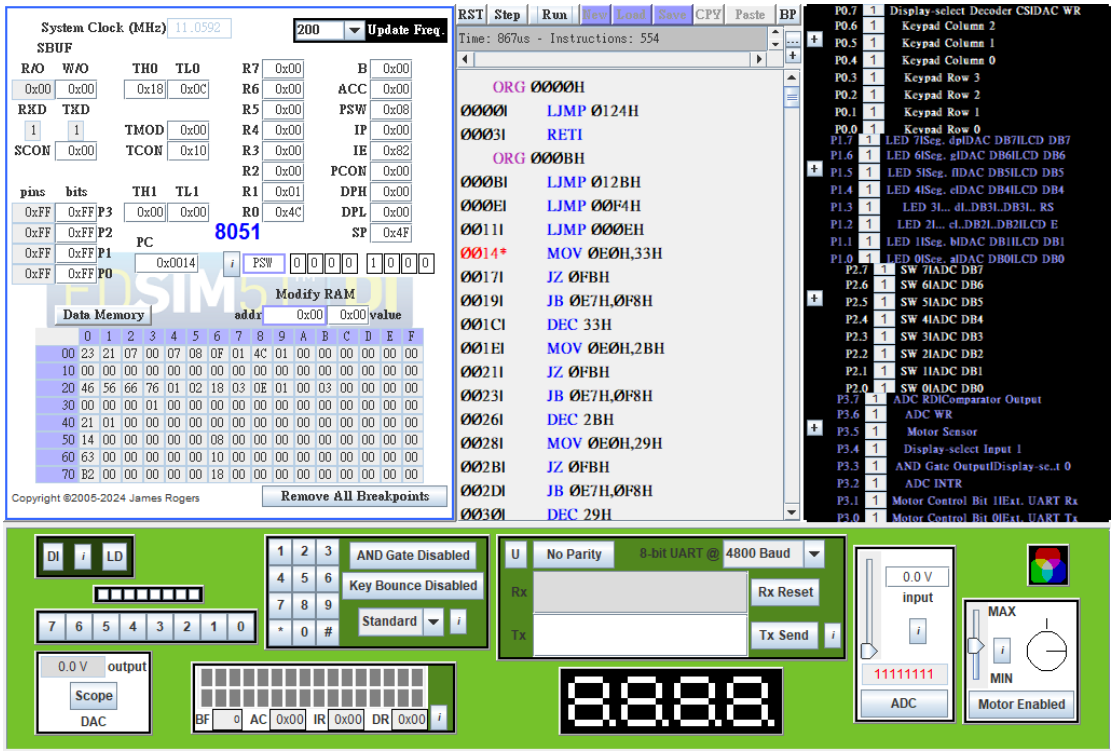
```
00000020  _T_SP (20~23, 1 address space each entry)   preemptive
00000024  _current_T                                  preemptive
00000025  _tmp0                                       preemptive
00000026  _tmp1                                       preemptive
00000027  _tmp2                                       preemptive
00000028  _bitmap                                     preemptive
00000029  _mutex  ┐ (semaphores to do                 test3threads
0000002A  _full   ├ classical bounded buffer)         test3threads
0000002B  _empty  ┘                                   test3threads
0000002C  _b_start                                    test3threads
0000002D  _b_end                                      test3threads
0000002E  _in_counter_1                               test3threads
0000002F  _in_counter_2                               test3threads
00000030  _buffer(30~32, 1 address space each entry)  test3threads
00000033  __1_turn ┐ (semaphores to make fair)        test3threads
00000034  __2_turn ┘                                  test3threads
```

And here's the function address map:

```
00000014  _Producer1                test3threads
00000063  _Producer2                test3threads
000000B2  _Consumer                 test3threads
000000F4  _main                     test3threads
00000124  __sdcc_gsinit_startup     test3threads
00000128  __mcs51_genRAMCLEAR       test3threads
00000129  __mcs51_genXINIT          test3threads
0000012A  __mcs51_genXRAMCLEAR      test3threads
0000012B  _timer0_ISR               test3threads
0000012F  _Bootstrap                preemptive
00000161  _ThreadCreate             preemptive
000001F3  _ThreadYield              preemptive
00000248  _ThreadExit               preemptive
000002A1  _myTimer0Handler          preemptive
```

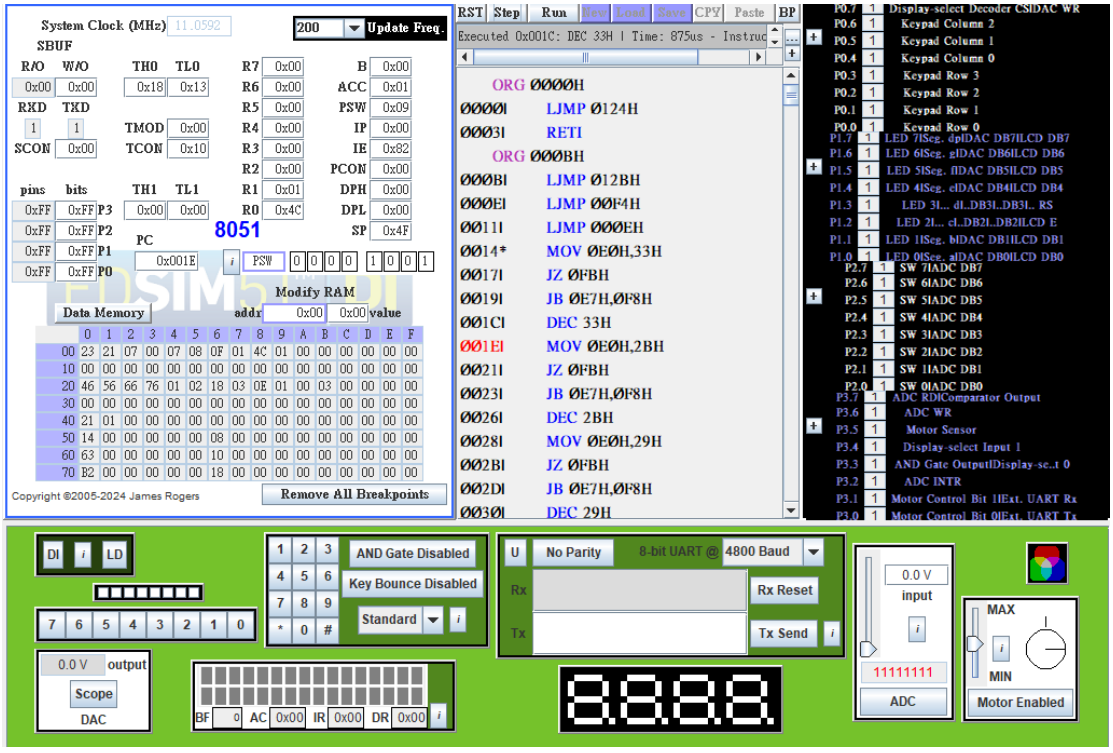
2. Screenshots and explanation:

- Producer1:



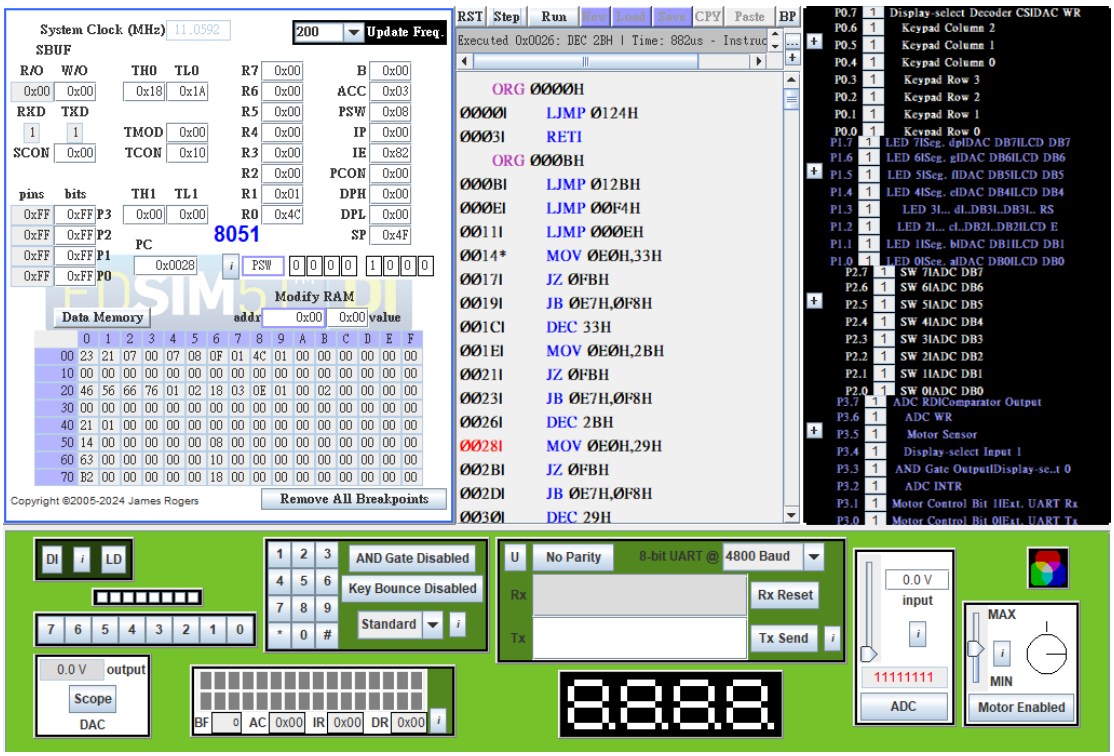
▲ Just enter Producer1

(mutex=1, full=0, empty=3, _1_turn=1, _2_turn=0).



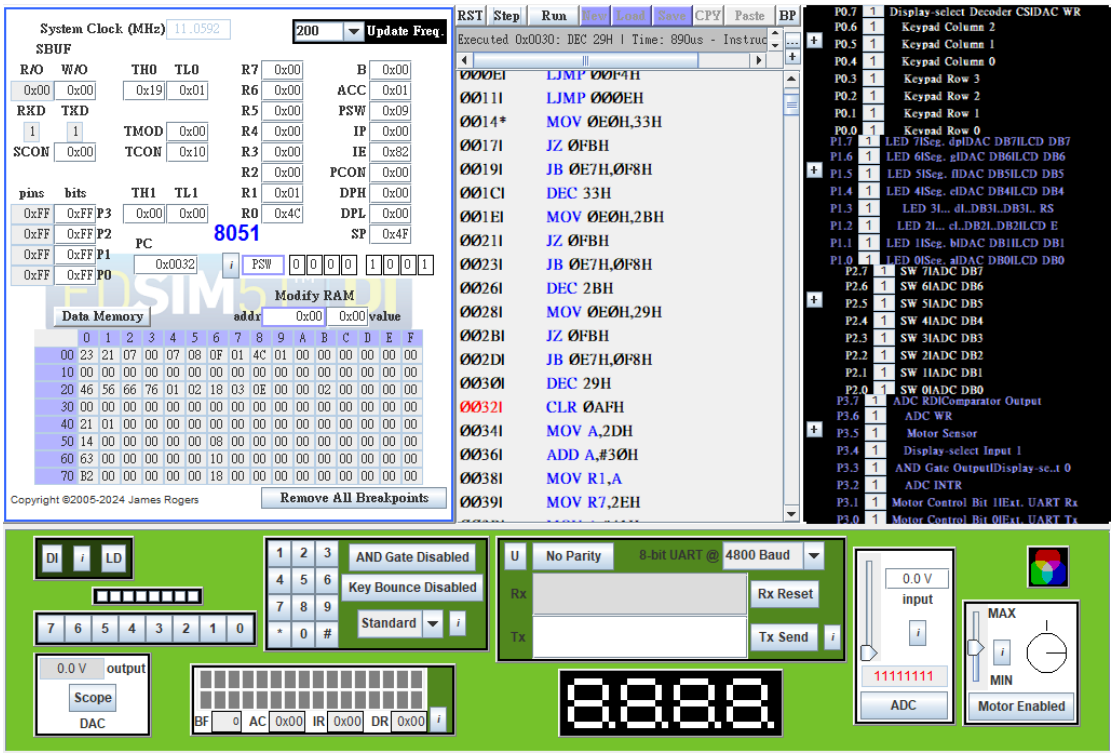
▲ After **`SemaphoreWait(_1_turn)`**

(mutex=1, full=0, empty=3, _1_turn=0, _2_turn=0).



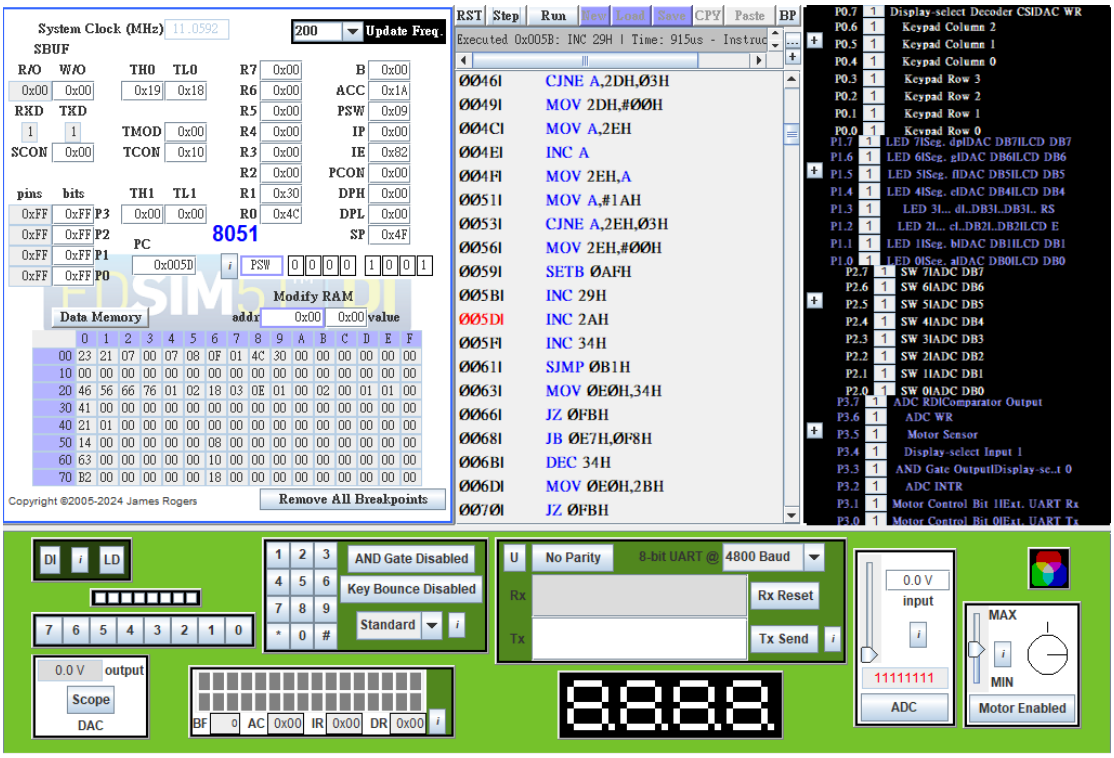
▲ After **`SemaphoreWait(empty)`**

(mutex=1, full=0, empty=2, _1_turn=0, _2_turn=0).



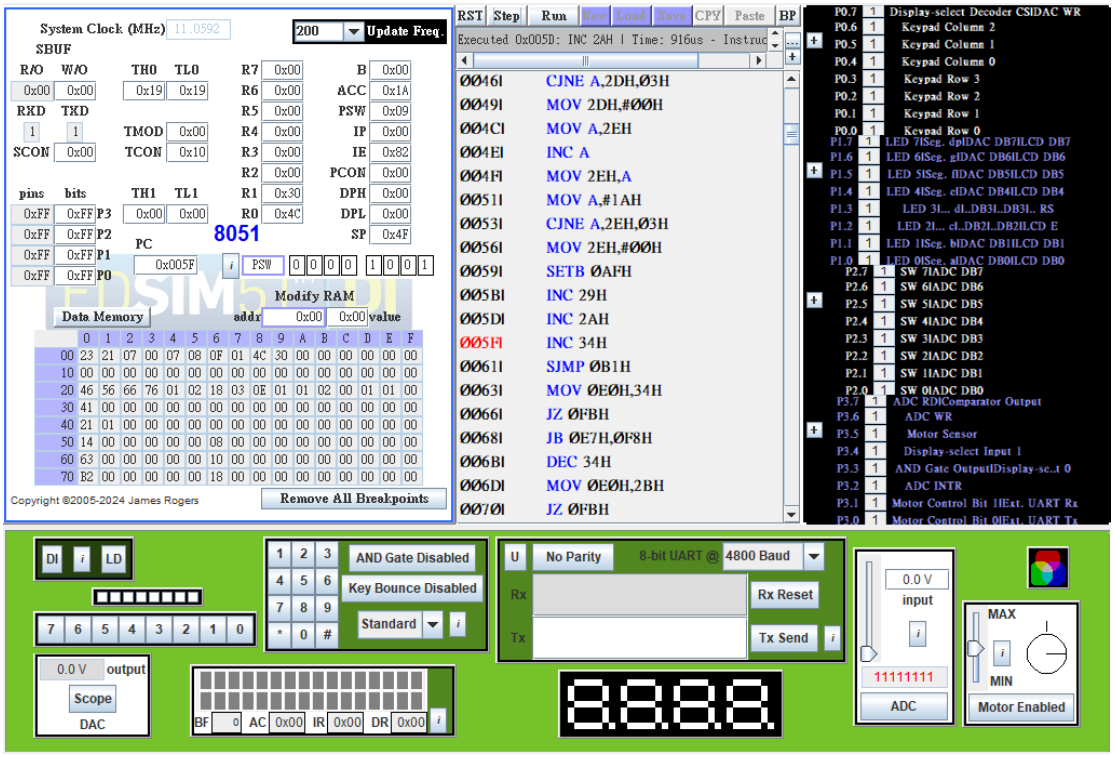
▲ After **`SemaphoreWait(mutex)`**

(mutex=0, full=0, empty=2, _1_turn=0, _2_turn=0).



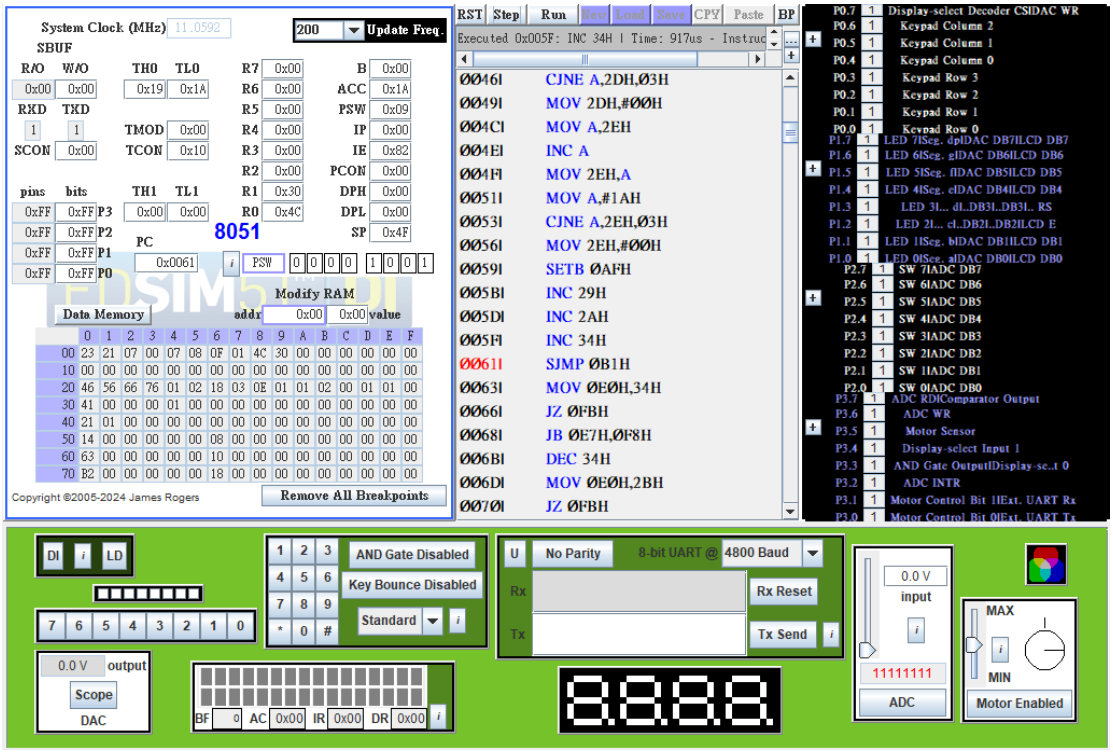
▲ After **`SemaphoreSignal(mutex)`**

(mutex=1, full=0, empty=2, _1_turn=0, _2_turn=0).



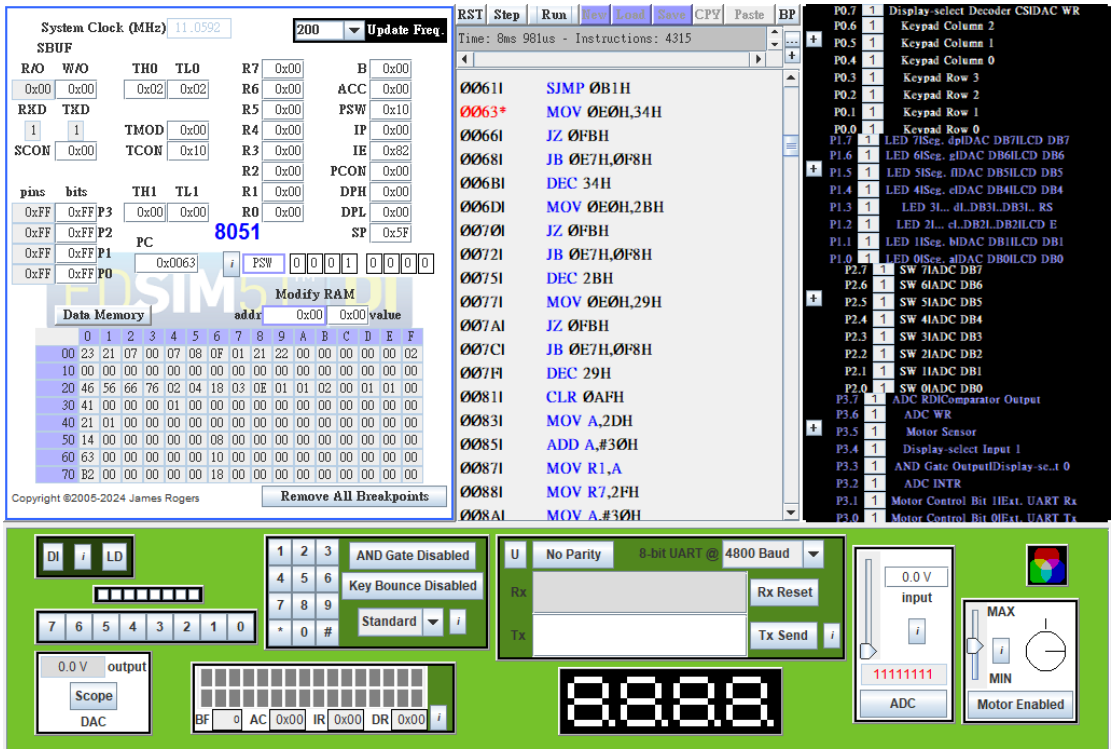
▲ After **`SemaphoreSignal(full)`**

(mutex=1, full=1, empty=2, _1_turn=0, _2_turn=0).



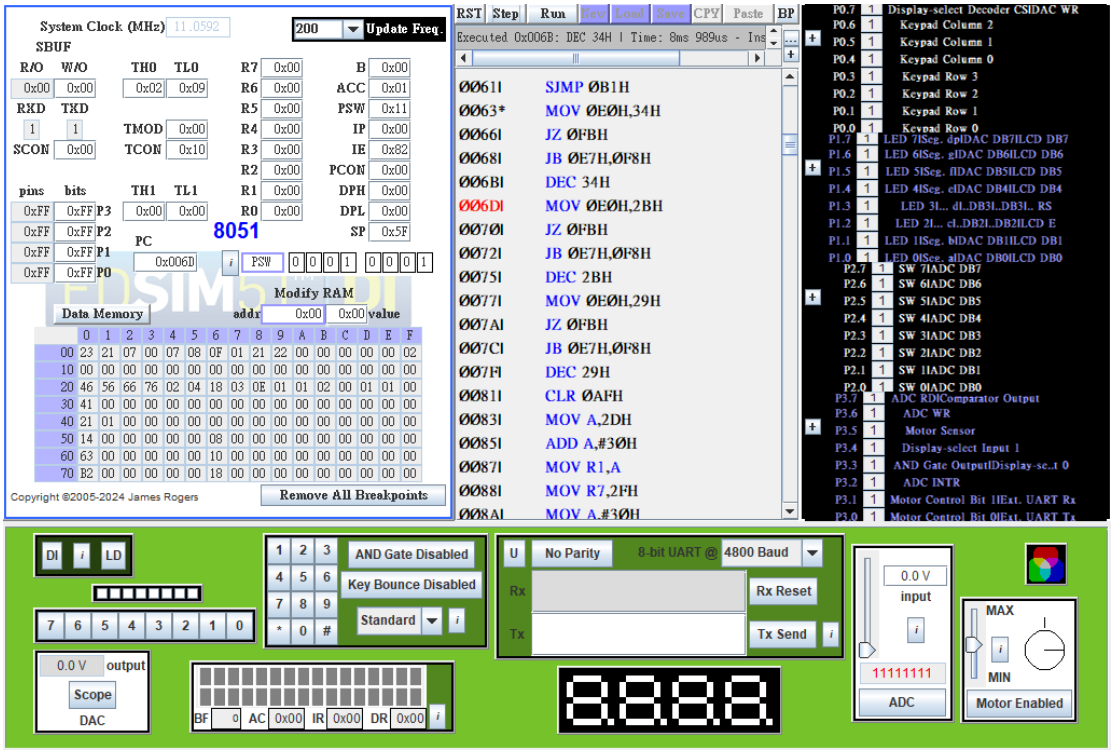
▲ After **`SemaphoreSignal(_2_turn)`**
(mutex=1, full=1, empty=2, _1_turn=0, _2_turn=1).

- Producer2:



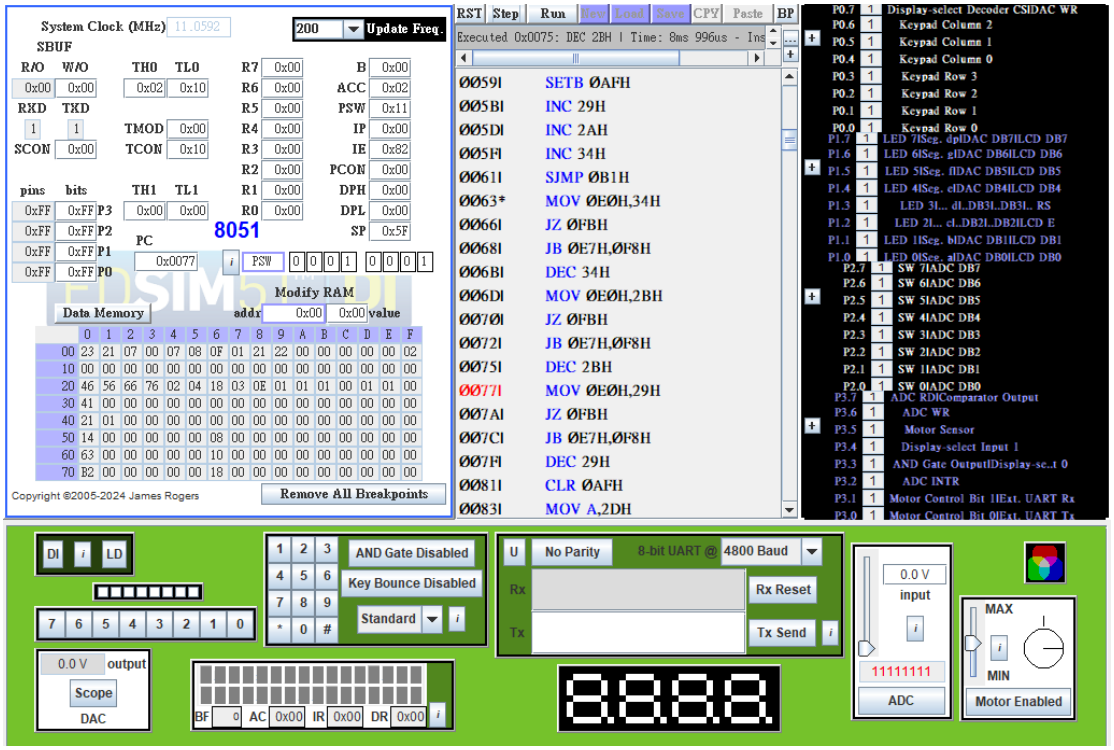
▲ Just enter Producer2

(mutex=1, full=1, empty=2, _1_turn=0, _2_turn=1).



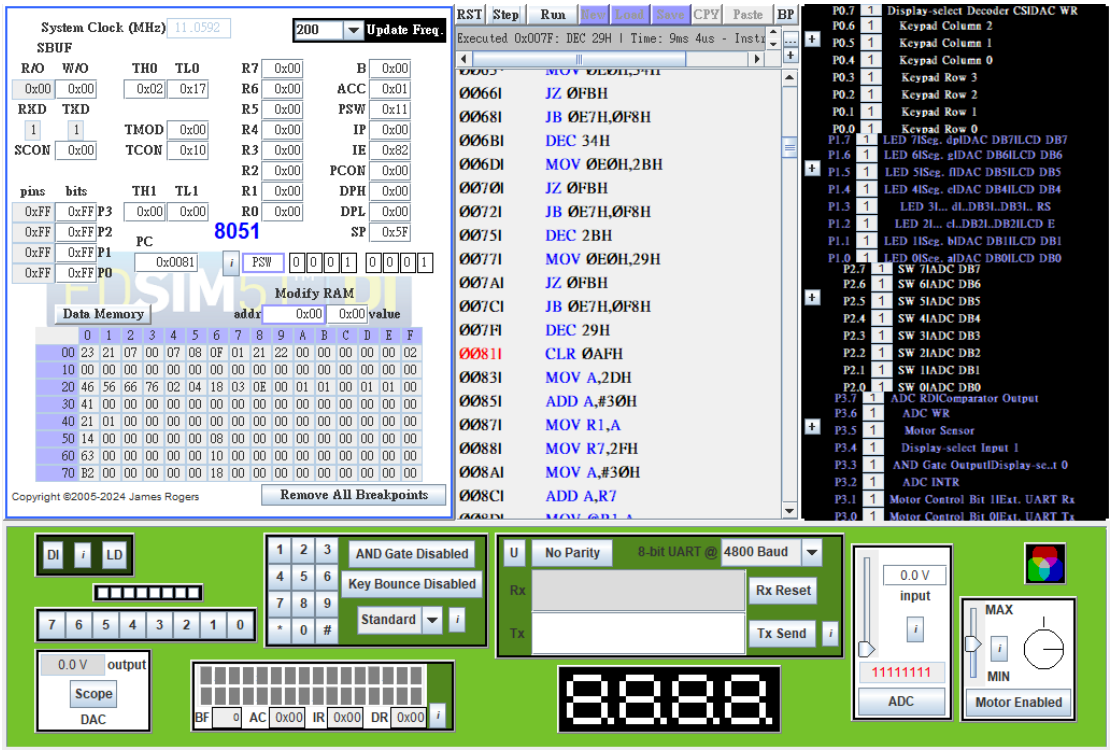
▲ After **`SemaphoreWait(_2_turn)`**

(mutex=1, full=1, empty=2, _1_turn=0, _2_turn=0).



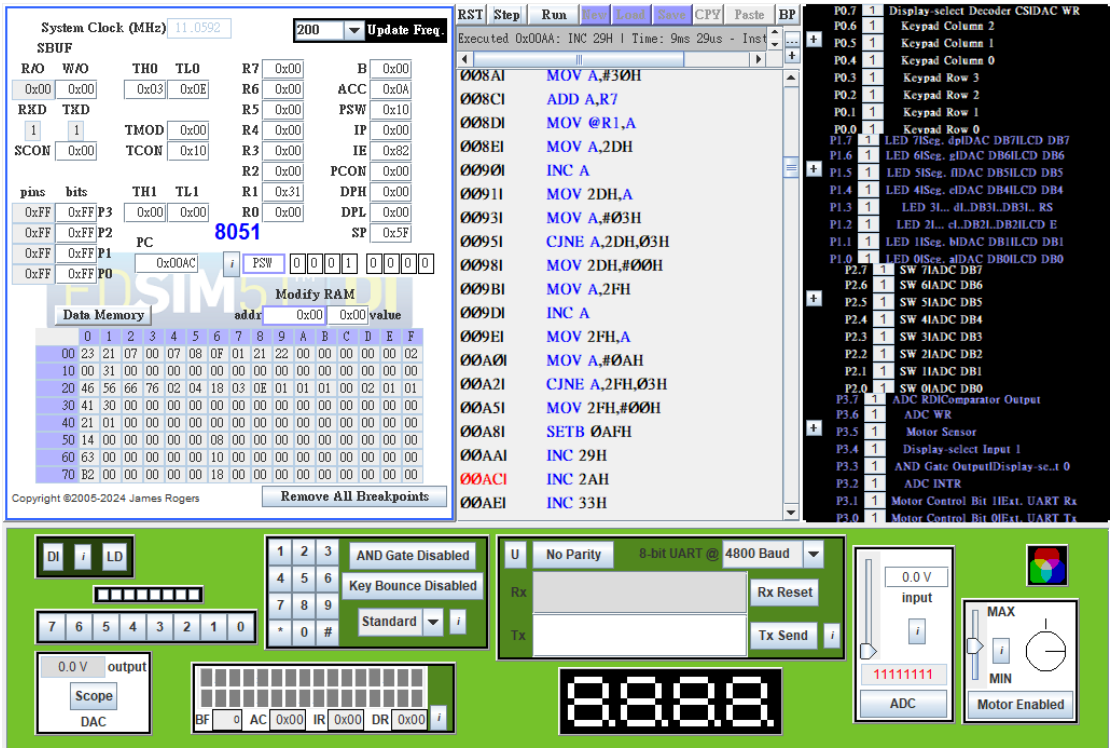
▲ After **`SemaphoreWait(empty)`**

(mutex=1, full=1, empty=1, _1_turn=0, _2_turn=0).



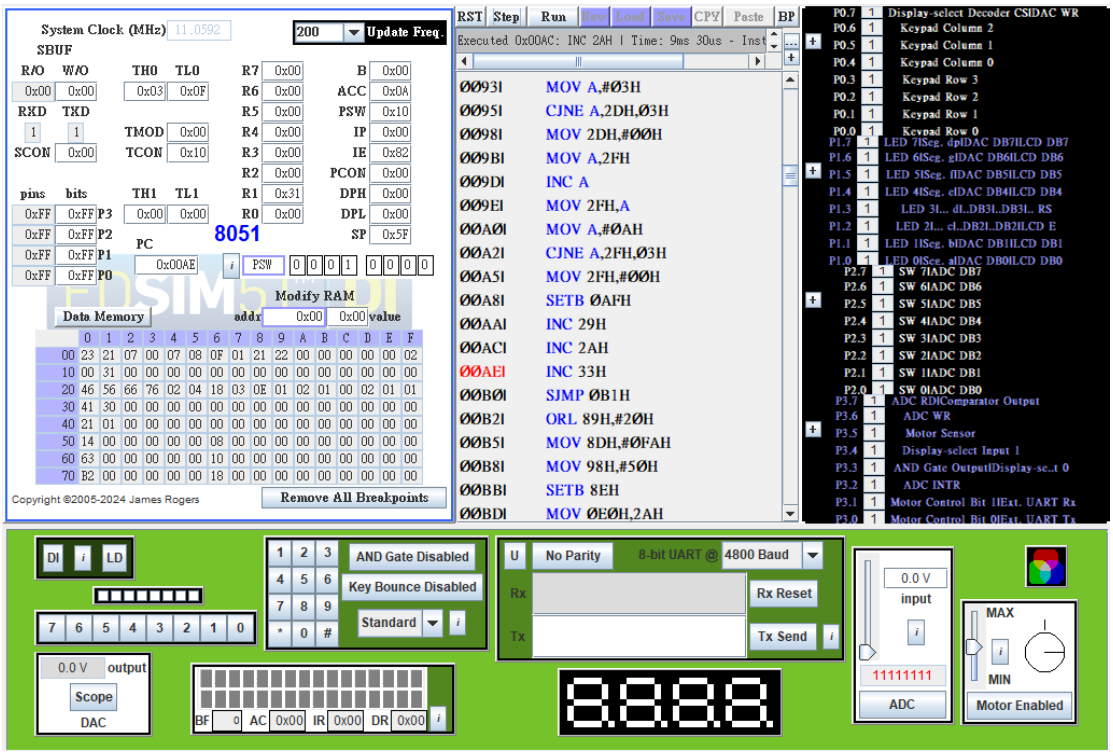
▲ After **`SemaphoreWait(mutex)`**

(mutex=0, full=1, empty=1, _1_turn=0, _2_turn=0).



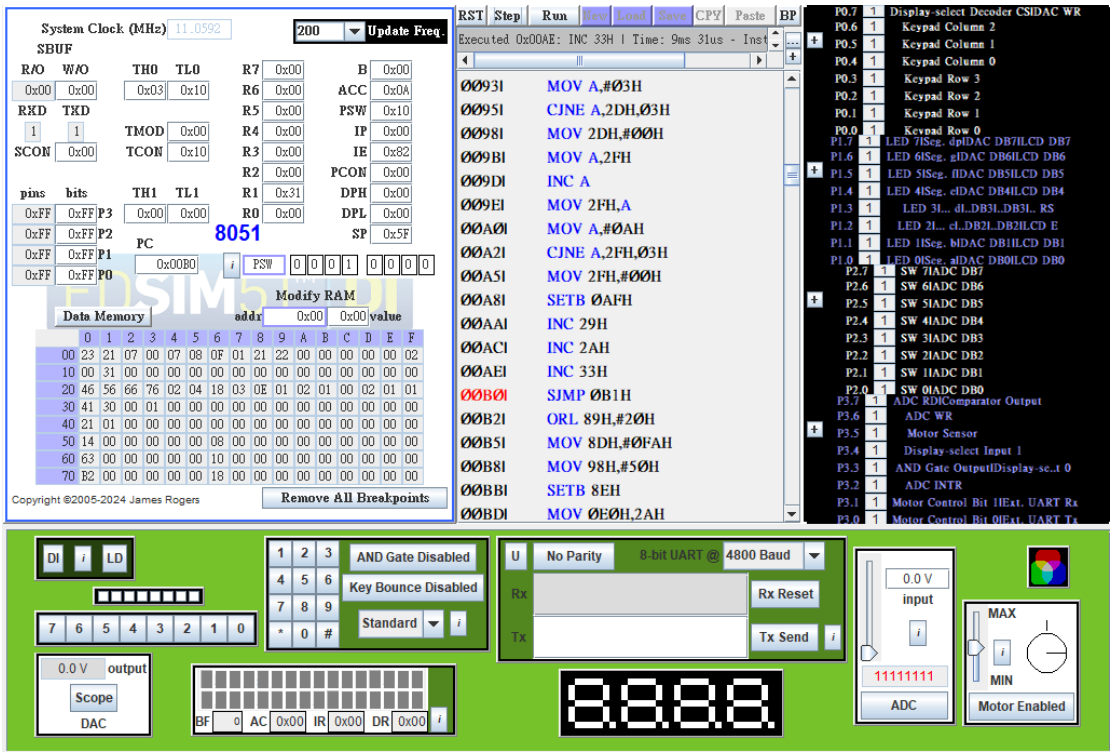
▲ After **`SemaphoreSignal(mutex)`**

(mutex=1, full=1, empty=1, _1_turn=0, _2_turn=0).



▲ After **`SemaphoreSignal(full)`**

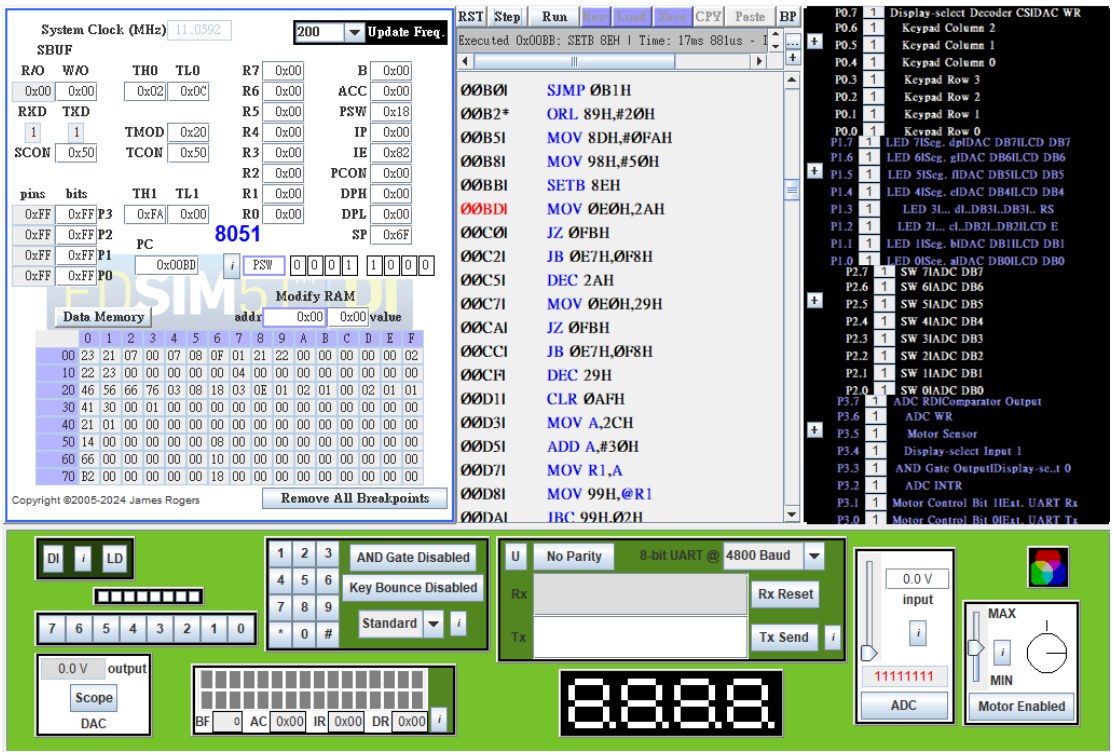
(mutex=1, full=2, empty=1, _1_turn=0, _2_turn=0).



▲ After **`SemaphoreSignal(_1_turn)`**

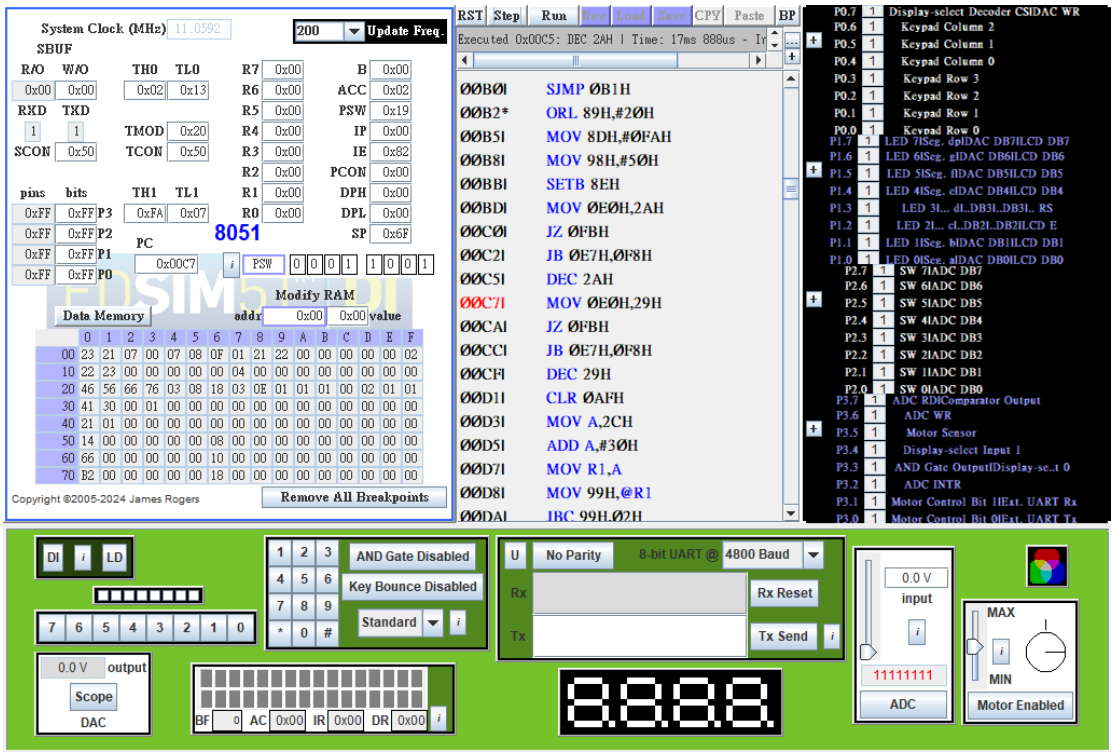
(mutex=1, full=2, empty=1, _1_turn=1, _2_turn=0).

- Consumer:



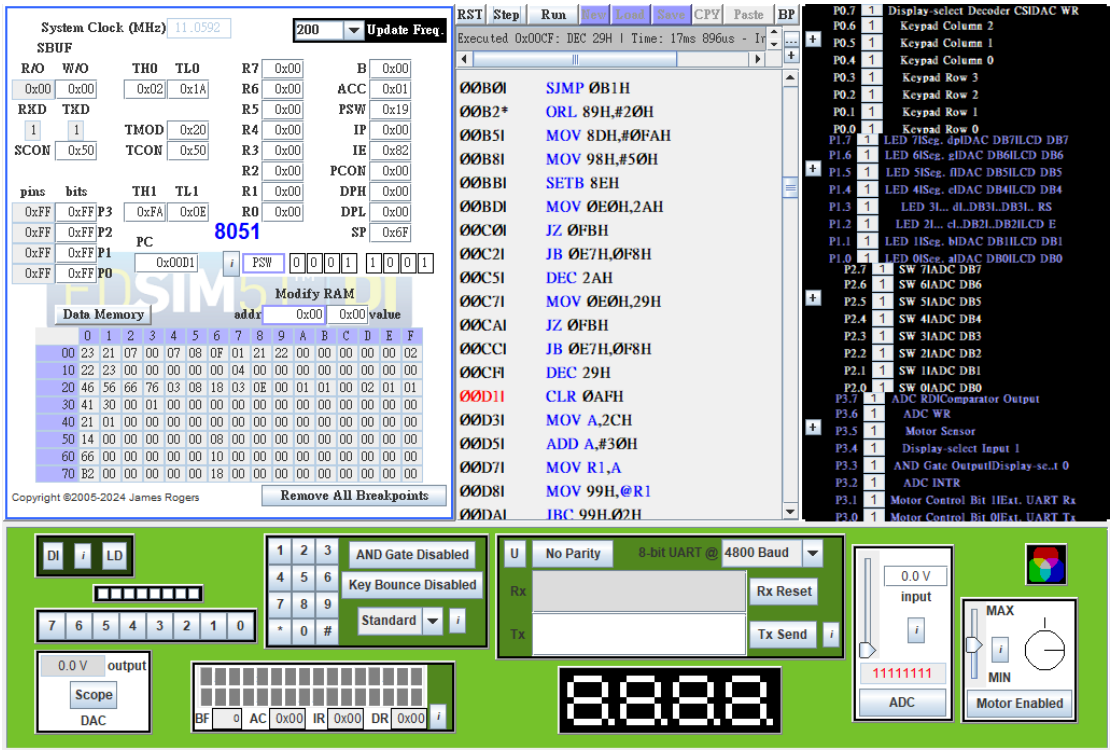
▲ Just enter Consumer, after finish UART init

(mutex=1, full=2, empty=1, _1_turn=1, _2_turn=0).



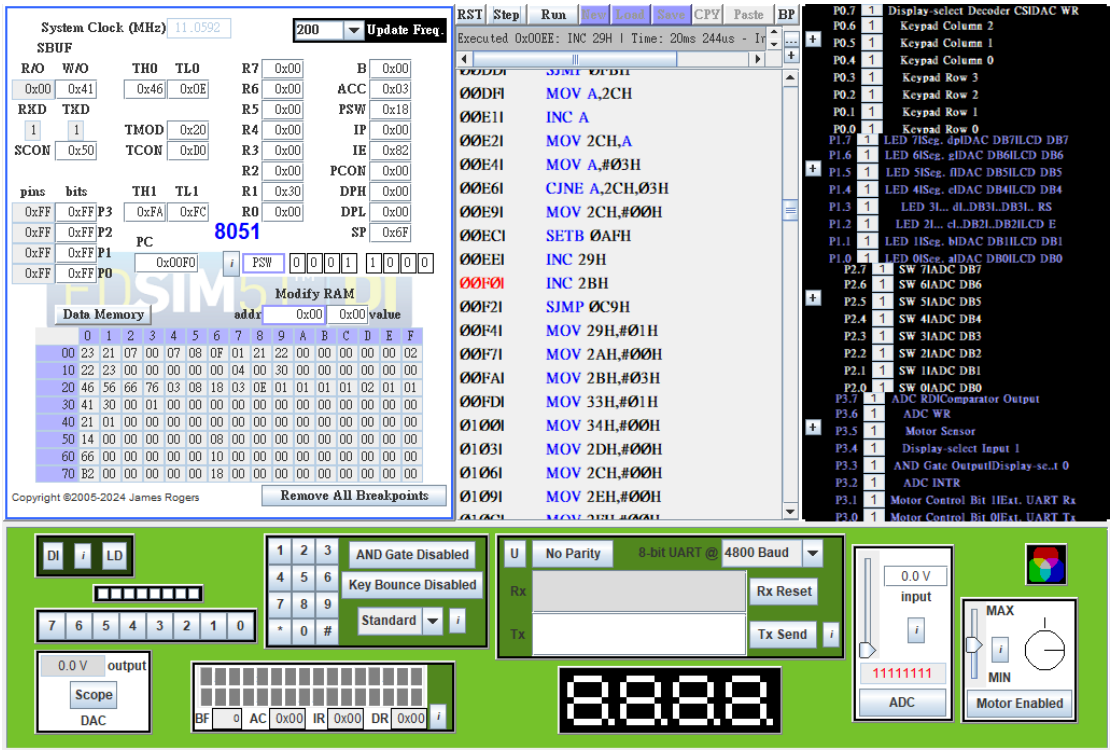
▲ After **`SemaphoreWait(full)`**

(mutex=1, full=1, empty=1, _1_turn=1, _2_turn=0).



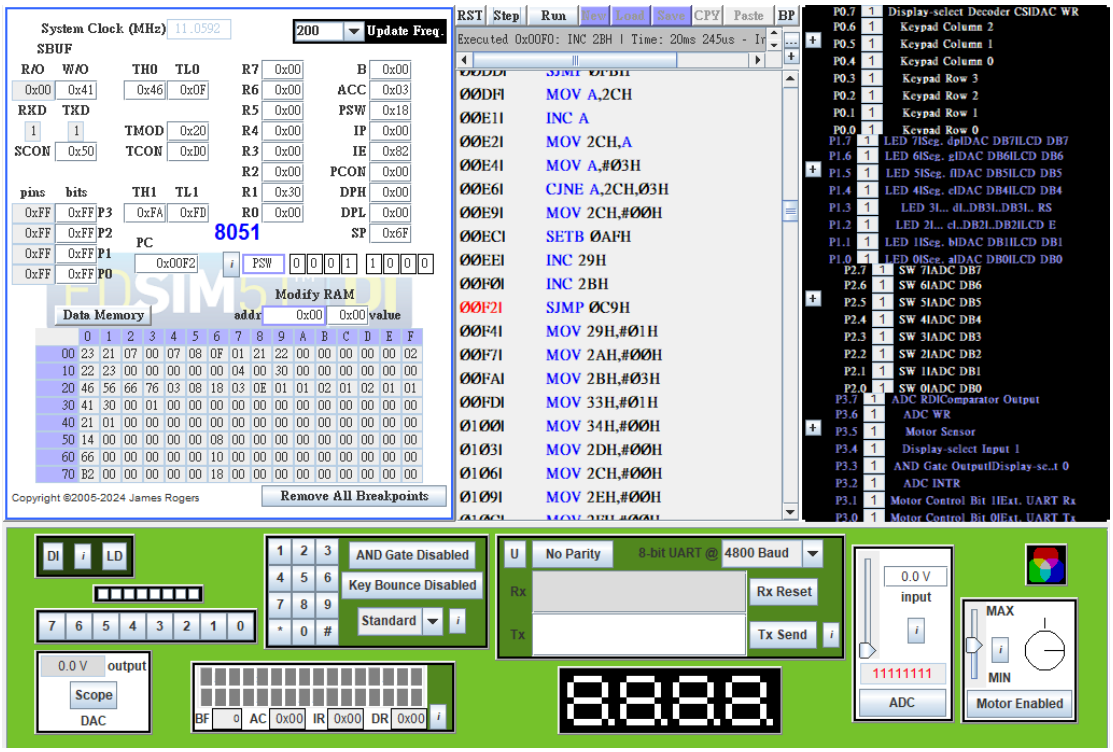
▲ After **`SemaphoreWait(mutex)`**

(mutex=0, full=1, empty=1, _1_turn=1, _2_turn=0).



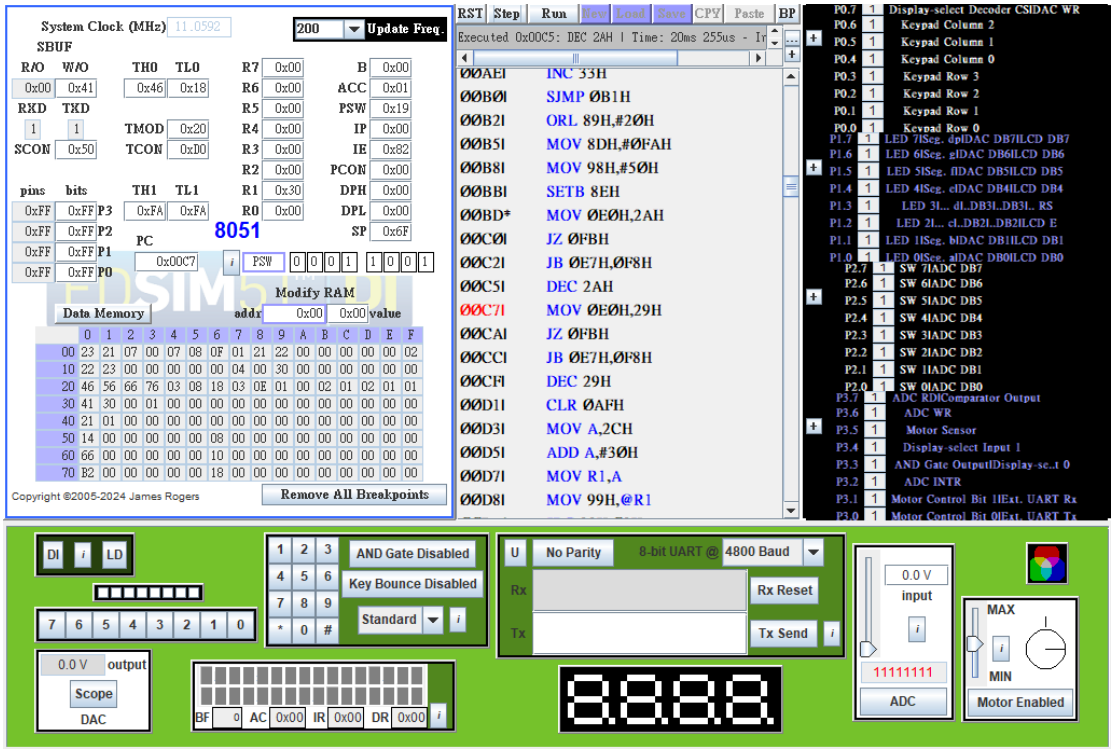
▲ After **`SemaphoreSignal(mutex)`**

(mutex=1, full=1, empty=1, _1_turn=1, _2_turn=0).



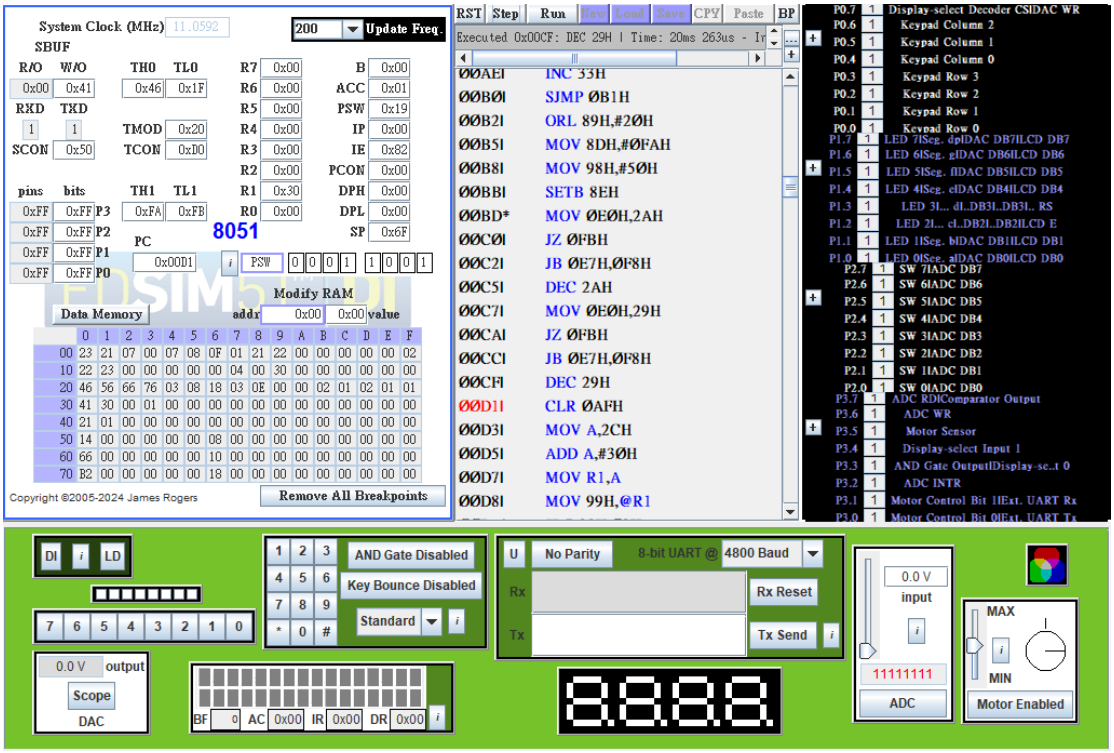
▲ After **`SemaphoreSignal(empty)`**

(mutex=1, full=1, empty=2, _1_turn=1, _2_turn=0).



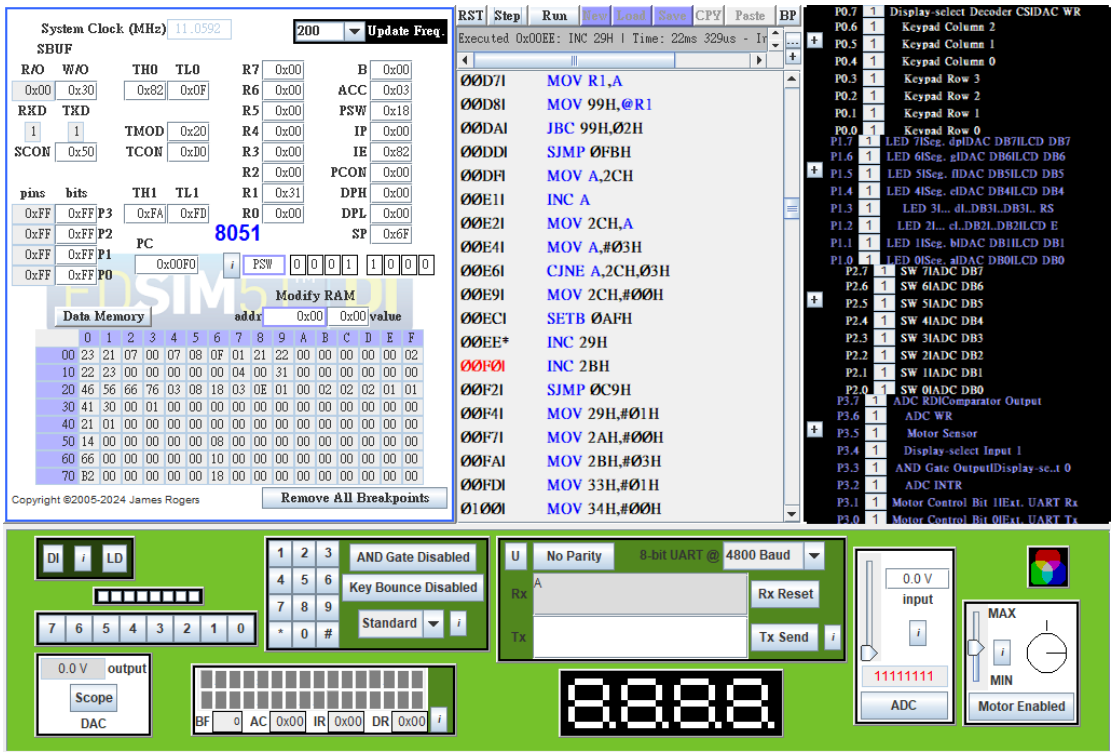
▲ After **`SemaphoreWait(full)`**

(mutex=1, full=0, empty=2, _1_turn=1, _2_turn=0).



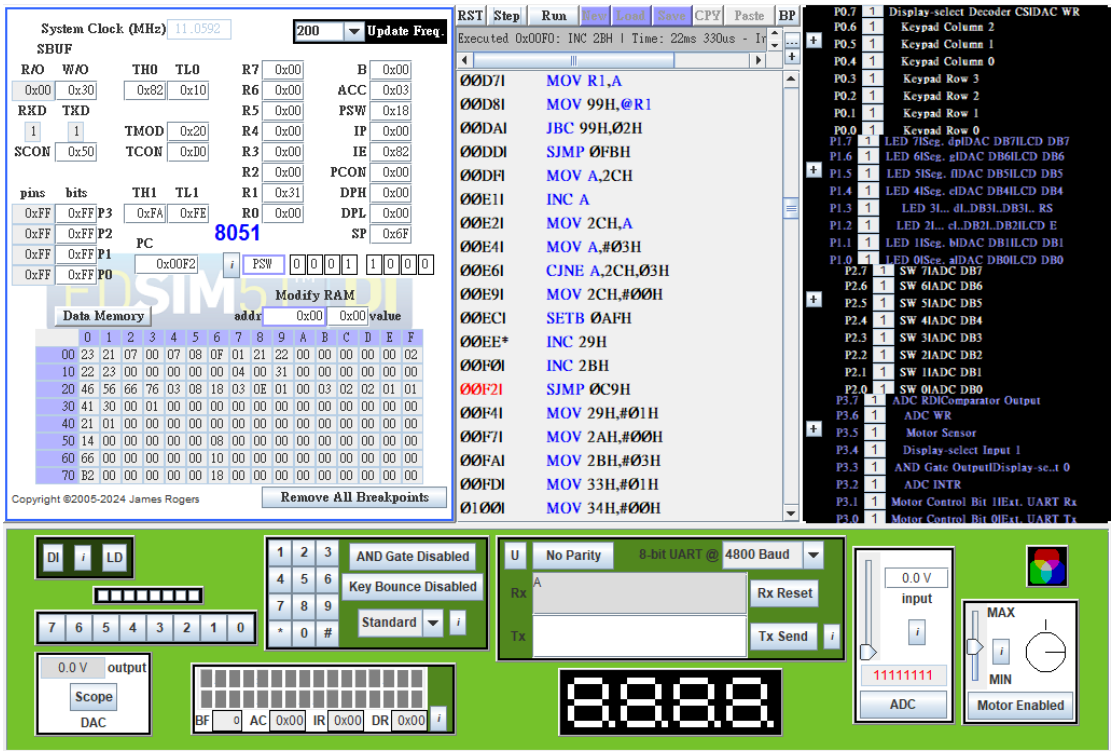
▲ After **`SemaphoreWait(mutex)`**

(mutex=0, full=0, empty=2, _1_turn=1, _2_turn=0).



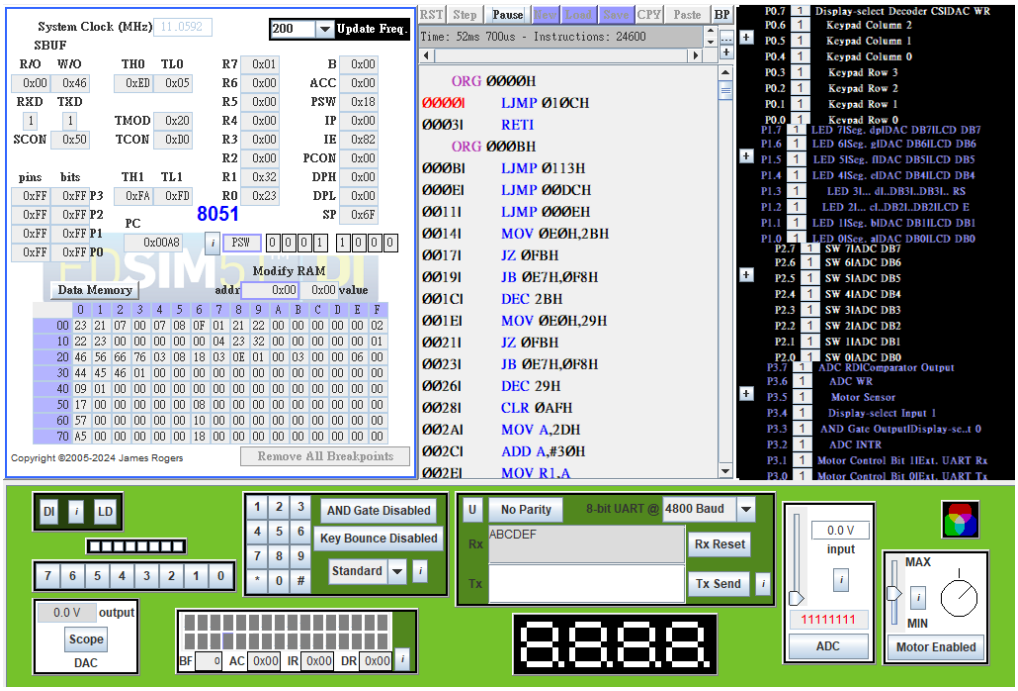
▲ After **`SemaphoreSignal(mutex)`**

(mutex=1, full=0, empty=2, _1_turn=1, _2_turn=0).

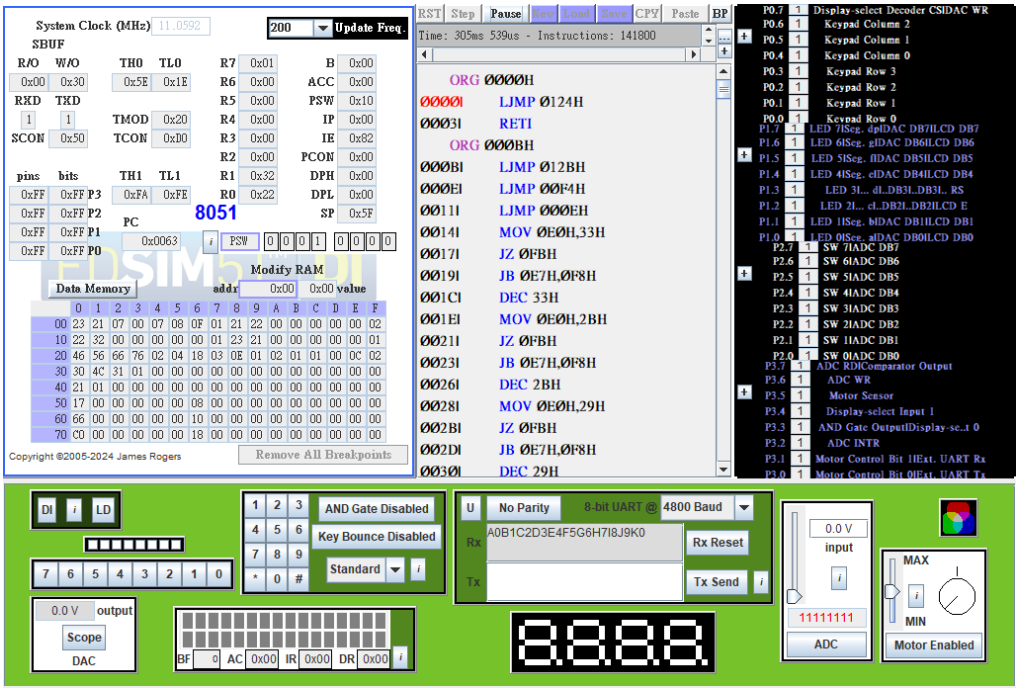


▲ After **`SemaphoreSignal(empty)`**
(mutex=1, full=0, empty=3, _1_turn=1, _2_turn=0).

- Unfair version:



- Fair version:



Difference of them is that in fair version, I add additional semaphores “_1_turn” and ” _2_turn” to indicate which producer should produce. For example, if it’s Producer1’s turn, then _1_turn=1 and _2_turn=0, and _2_turn will be signaled if and only if Producer1 finish, at the same time, _1_turn will be set to 0, which means it’s time for Producer2 to work.