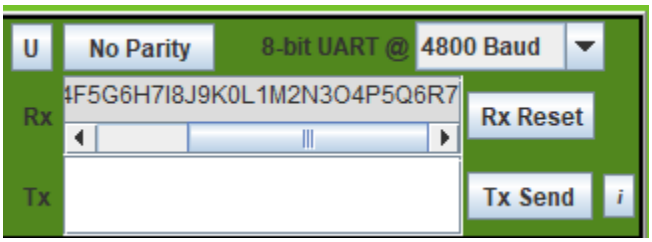
Operating System CP4

許木羽 / 111000177

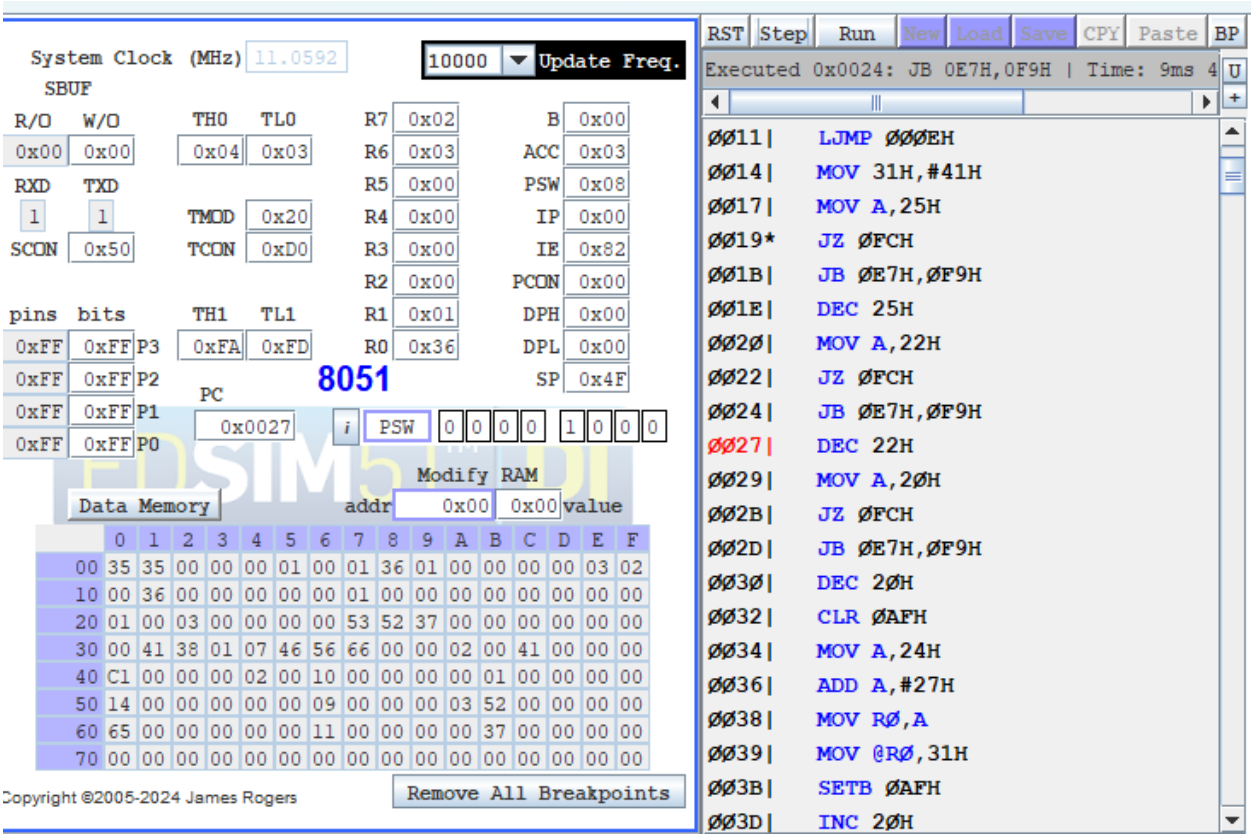


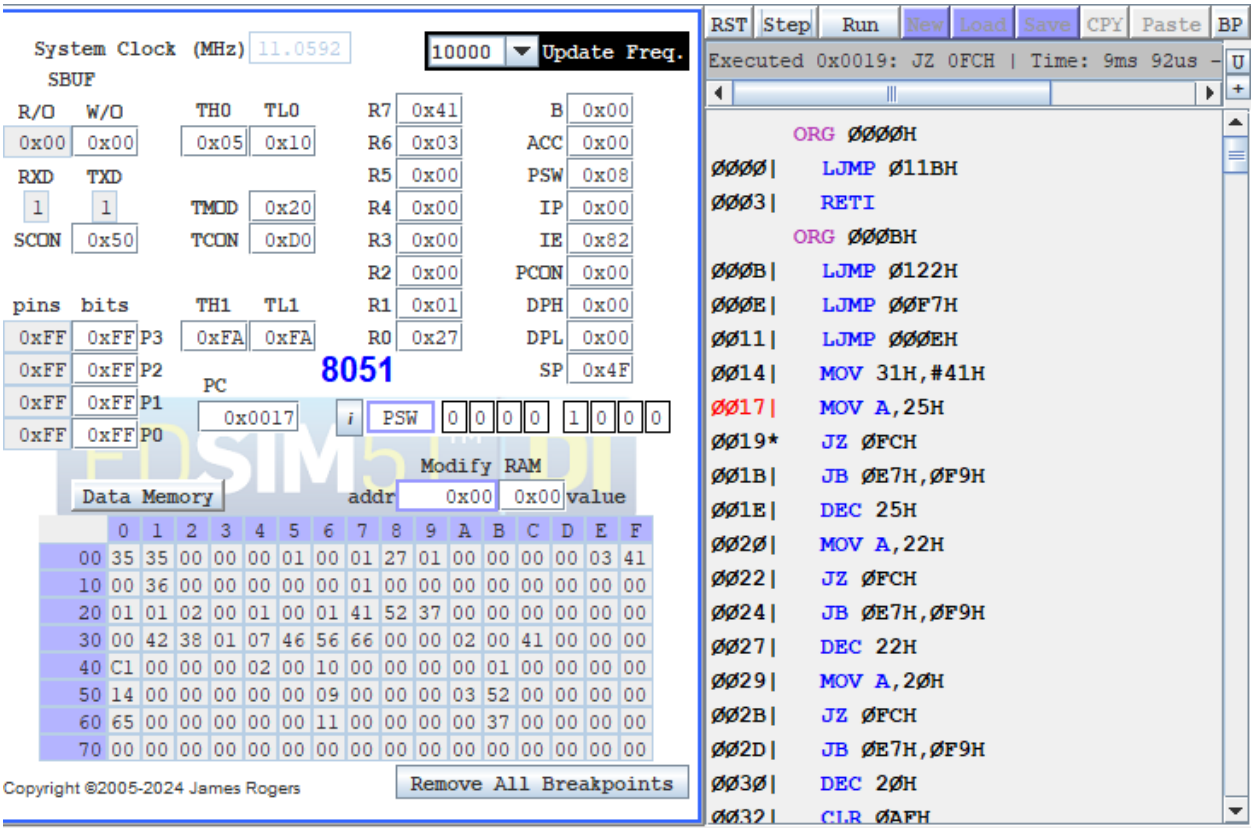
The semaphore like CP3, however there is an extra semaphore variable to take turn for the producer. Here as one producer output ABCDEFG..., and the other producer output 01234..., my fairness system is to output everything in turn, so the expected output will be:

A0B1C2D3E4F5...

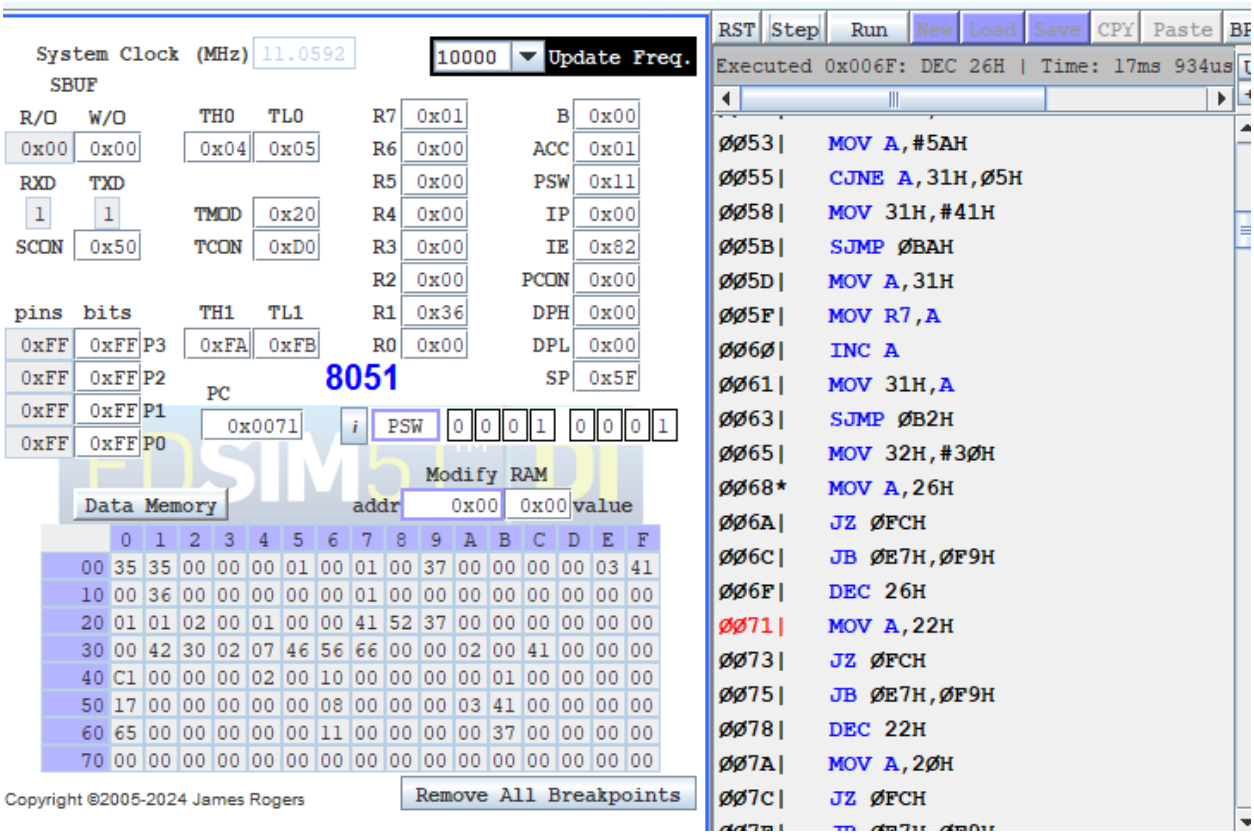
There will be new 2 semaphore variable, let say variable A and B. For producer 1, it needs to wait the signal from A and then after finishing sending signal B. Producer 2 will wait for the signal B and then after finish, send the signal A. Therefore, the turn will be correct.

```
      Value  Global                                 Global Defined In Module
      -----  -------------------------------   ------------------------
C:    00000014  _Producer1                          test3preempt
C:    00000065  _Producer2                          test3preempt
C:    000000B6  _Consumer                           test3preempt
C:    000000F7  _main                               test3preempt
C:    0000011B  __sdcc_gsinit_startup               test3preempt
C:    0000011F  __mcs51_genRAMCLEAR                 test3preempt
C:    00000120  __mcs51_genXINIT                    test3preempt
C:    00000121  __mcs51_genXRAMCLEAR                test3preempt
C:    00000122  _timer0_ISR                         test3preempt
C:    00000126  _Bootstrap                          preemptive
C:    0000014C  _ThreadCreate                       preemptive
C:    000001C9  _ThreadYield                        preemptive
C:    00000223  _ThreadExit                         preemptive
C:    00000239  _myTimer0Handler                    preemptive
```


System Clock (MHz) 11.0592
10000 Update Freq.
SBUF
R/O W/O TH0 TL0 R7 0x02 B 0x00
0x00 0x00 0x04 0x03 R6 0x03 ACC 0x03
RXD TXD R5 0x00 PSW 0x08
1 1 TMOD 0x20 R4 0x00 IP 0x00
SCON 0x50 TCON 0xD0 R3 0x00 IE 0x82
R2 0x00 PCON 0x00
pins bits TH1 TL1 R1 0x01 DPH 0x00
0xFF 0xFF P3 0xFA 0xFD R0 0x36 DPL 0x00
0xFF 0xFF P2 8051 SP 0x4F
PC
0xFF 0xFF P1 0x0027 i PSW 0 0 0 0 1 0 0 0
0xFF 0xFF P0
Modify RAM
Data Memory addr 0x00 0x00 value
0 1 2 3 4 5 6 7 8 9 A B C D E F
00 35 35 00 00 00 01 00 01 36 01 00 00 00 00 03 02
10 00 36 00 00 00 00 00 01 00 00 00 00 00 00 00 00
20 01 00 03 00 00 00 00 53 52 37 00 00 00 00 00 00
30 00 41 38 01 07 46 56 66 00 00 02 00 41 00 00 00
40 C1 00 00 00 02 00 10 00 00 00 00 01 00 00 00 00
50 14 00 00 00 00 00 09 00 00 00 03 52 00 00 00 00
60 65 00 00 00 00 00 11 00 00 00 00 37 00 00 00 00
70 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Copyright ©2005-2024 James Rogers
Remove All Breakpoints
RST Step Run New Load Save CPY Paste BP
Executed 0x0024: JB 0E7H,0F9H | Time: 9ms 4
ØØ11| LJMP ØØØEH
ØØ14| MOV 31H,#41H
ØØ17| MOV A,25H
ØØ19* JZ ØFCH
ØØ1B| JB ØE7H,ØF9H
ØØ1E| DEC 25H
ØØ2Ø| MOV A,22H
ØØ22| JZ ØFCH
ØØ24| JB ØE7H,ØF9H
ØØ27| DEC 22H
ØØ29| MOV A,2ØH
ØØ2B| JZ ØFCH
ØØ2D| JB ØE7H,ØF9H
ØØ3Ø| DEC 2ØH
ØØ32| CLR ØAFH
ØØ34| MOV A,24H
ØØ36| ADD A,#27H
ØØ38| MOV RØ,A
ØØ39| MOV @RØ,31H
ØØ3B| SETB ØAFH
ØØ3D| INC 2ØH

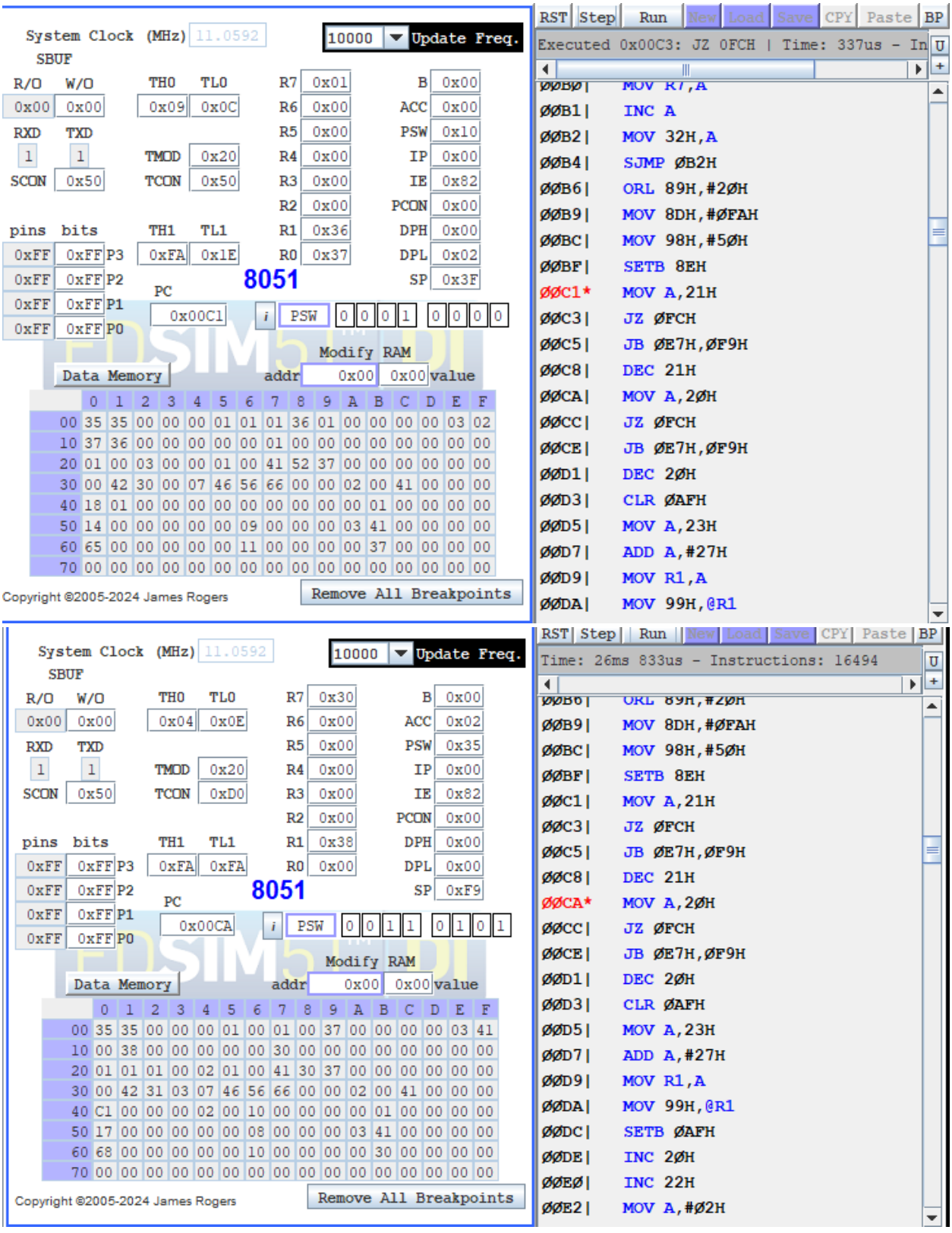

System Clock (MHz) 11.0592
10000 Update Freq.
SBUF
R/O W/O TH0 TL0 R7 0x41 B 0x00
0x00 0x00 0x05 0x10 R6 0x03 ACC 0x00
RXD TXD R5 0x00 PSW 0x08
1 1 TMOD 0x20 R4 0x00 IP 0x00
SCON 0x50 TCON 0xD0 R3 0x00 IE 0x82
R2 0x00 PCON 0x00
pins bits TH1 TL1 R1 0x01 DPH 0x00
0xFF 0xFF P3 0xFA 0xFA R0 0x27 DPL 0x00
0xFF 0xFF P2 8051 SP 0x4F
PC
0xFF 0xFF P1 0x0017 i PSW 0 0 0 0 1 0 0 0
0xFF 0xFF P0
Modify RAM
Data Memory addr 0x00 0x00 value
0 1 2 3 4 5 6 7 8 9 A B C D E F
00 35 35 00 00 00 01 00 01 27 01 00 00 00 00 03 41
10 00 36 00 00 00 00 00 01 00 00 00 00 00 00 00 00
20 01 01 02 00 01 00 01 41 52 37 00 00 00 00 00 00
30 00 42 38 01 07 46 56 66 00 00 02 00 41 00 00 00
40 C1 00 00 00 02 00 10 00 00 00 00 01 00 00 00 00
50 14 00 00 00 00 00 09 00 00 00 03 52 00 00 00 00
60 65 00 00 00 00 00 11 00 00 00 00 37 00 00 00 00
70 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Copyright ©2005-2024 James Rogers
Remove All Breakpoints
RST Step Run New Load Save CPY Paste BP
Executed 0x0019: JZ 0FCH | Time: 9ms 92us -
ORG ØØØØH
ØØØØ| LJMP Ø11BH
ØØØ3| RETI
ORG ØØØBH
ØØØB| LJMP Ø122H
ØØØE| LJMP ØØF7H
ØØ11| LJMP ØØØEH
ØØ14| MOV 31H,#41H
ØØ17| MOV A,25H
ØØ19* JZ ØFCH
ØØ1B| JB ØE7H,ØF9H
ØØ1E| DEC 25H
ØØ2Ø| MOV A,22H
ØØ22| JZ ØFCH
ØØ24| JB ØE7H,ØF9H
ØØ27| DEC 22H
ØØ29| MOV A,2ØH
ØØ2B| JZ ØFCH
ØØ2D| JB ØE7H,ØF9H
ØØ3Ø| DEC 2ØH
ØØ32| CLR ØAFH

Above image is producer 1 first iteration and below is the second iteration. From the image, we can see that the first will not stall while the second will stall. This is because the producer 1 already did the output and haven't wait for the producer 2 to output. After some stalling it will go do the producer 2 first as below image

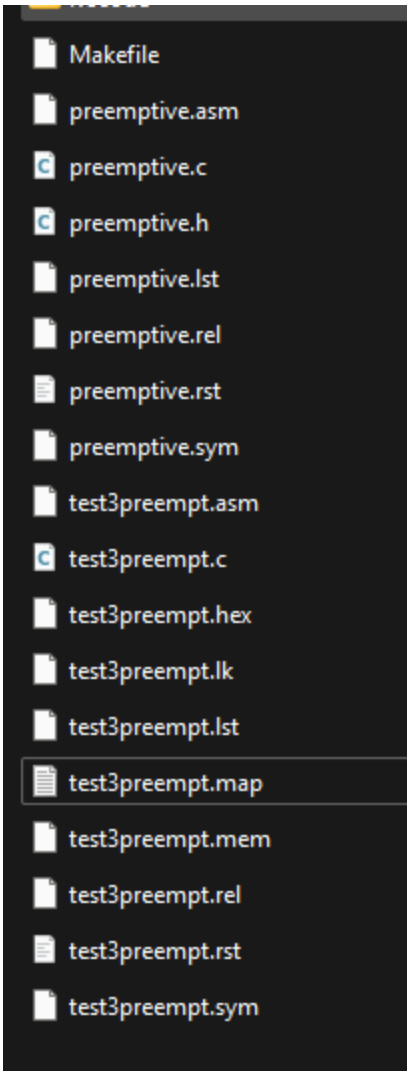


	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	35	35	00	00	00	01	00	01	00	37	00	00	00	00	03	41
10	00	36	00	00	00	00	00	01	00	00	00	00	00	00	00	00
20	01	01	02	00	01	00	00	41	52	37	00	00	00	00	00	00
30	00	42	30	02	07	46	56	66	00	00	02	00	41	00	00	00
40	C1	00	00	00	02	00	10	00	00	00	00	01	00	00	00	00
50	17	00	00	00	00	00	08	00	00	00	03	41	00	00	00	00
60	65	00	00	00	00	00	11	00	00	00	00	37	00	00	00	00
70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Below is the consumer



First image is the consumer get stalled, and below is the consumer succeed from the full signal.


Makefile
preemptive.asm
preemptive.c
preemptive.h
preemptive.lst
preemptive.rel
preemptive.rst
preemptive.sym
test3preempt.asm
test3preempt.c
test3preempt.hex
test3preempt.lk
test3preempt.lst
test3preempt.map
test3preempt.mem
test3preempt.rel
test3preempt.rst
test3preempt.sym