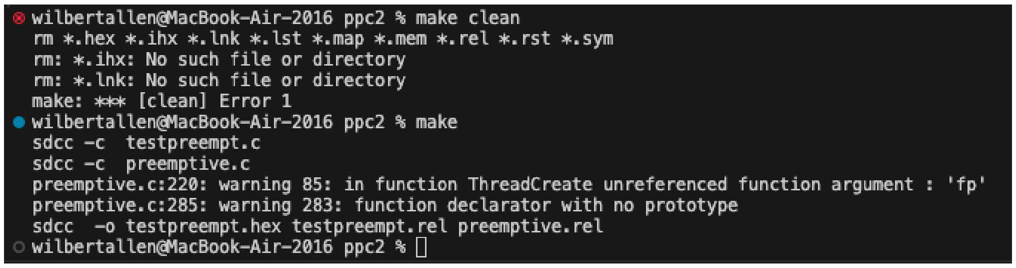
鍾維聖 110062261

Programming Project Checkpoint 2 Report

Typescript for Compiling



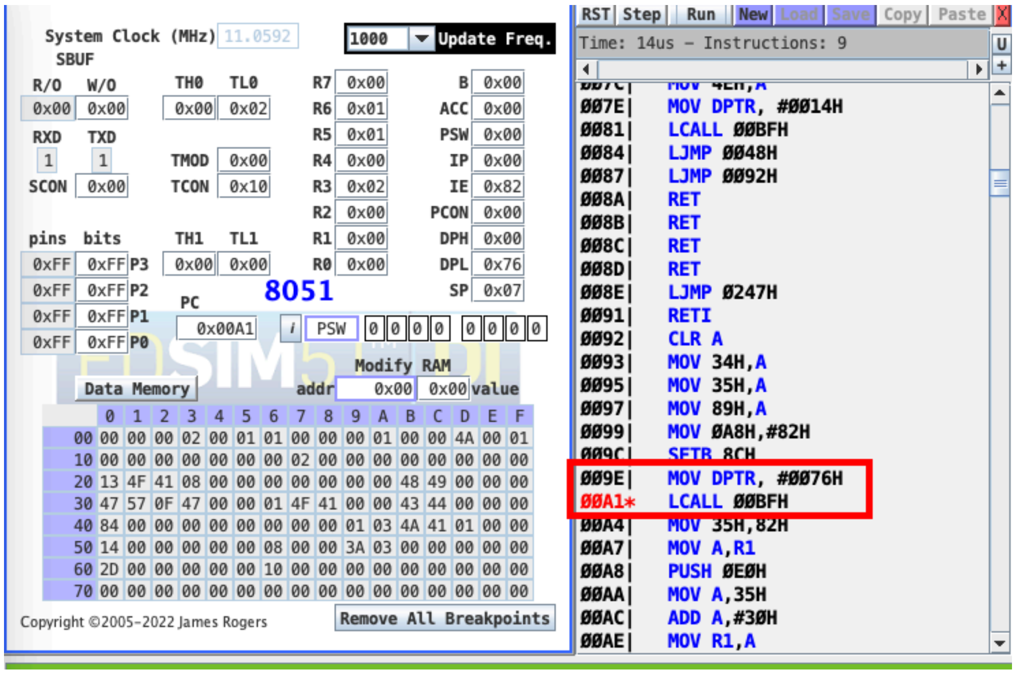
```
wilbertallen@MacBook-Air-2016 ppc2 % make clean
rm *.hex *.ihx *.lnk *.lst *.map *.mem *.rel *.rst *.sym
rm: *.ihx: No such file or directory
rm: *.lnk: No such file or directory
make: *** [clean] Error 1
wilbertallen@MacBook-Air-2016 ppc2 % make
sdcc -c  testpreempt.c
sdcc -c  preemptive.c
preemptive.c:220: warning 85: in function ThreadCreate unreferenced function argument : 'fp'
preemptive.c:285: warning 283: function declarator with no prototype
sdcc  -o testpreempt.hex testpreempt.rel preemptive.rel
wilbertallen@MacBook-Air-2016 ppc2 %
```

Fig.1 Typescript for compiling using the given makefile

Before Each ThreadCreate Call

Based on my understanding, ThreadCreate is called 2 times, one is for ThreadCreate(main), and the other is for ThreadCreate(Producer)

ThreadCreate(main);



```
System Clock (MHz) 11.0592        1000  Update Freq.
SBUF
R/O   W/O      TH0   TL0     R7 0x00       B 0x00
0x00  0x00     0x00  0x02    R6 0x01     ACC 0x00
RXD   TXD                    R5 0x01     PSW 0x00
1     1        TMOD  0x00    R4 0x00      IP 0x00
SCON  0x00     TCON  0x10    R3 0x02      IE 0x82
                             R2 0x00    PCON 0x00
pins  bits     TH1   TL1     R1 0x00     DPH 0x00
0xFF  0xFF P3  0x00  0x00    R0 0x00     DPL 0x76
0xFF  0xFF P2                             SP 0x07
0xFF  0xFF P1  PC 0x00A1   PSW 0 0 0 0  0 0 0 0
0xFF  0xFF P0
Modify RAM  addr 0x00  0x00 value
```

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	00	00	00	02	00	01	01	00	00	00	01	00	00	4A	00	01
10	00	00	00	00	00	00	00	02	00	00	00	00	00	00	00	00
20	13	4F	41	08	00	00	00	00	00	00	00	48	49	00	00	00
30	47	57	0F	47	00	00	01	4F	41	00	00	43	44	00	00	00
40	84	00	00	00	00	00	00	00	00	01	03	4A	41	01	00	00
50	14	00	00	00	00	00	08	00	00	3A	03	00	00	00	00	00
60	2D	00	00	00	00	00	10	00	00	00	00	00	00	00	00	00
70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

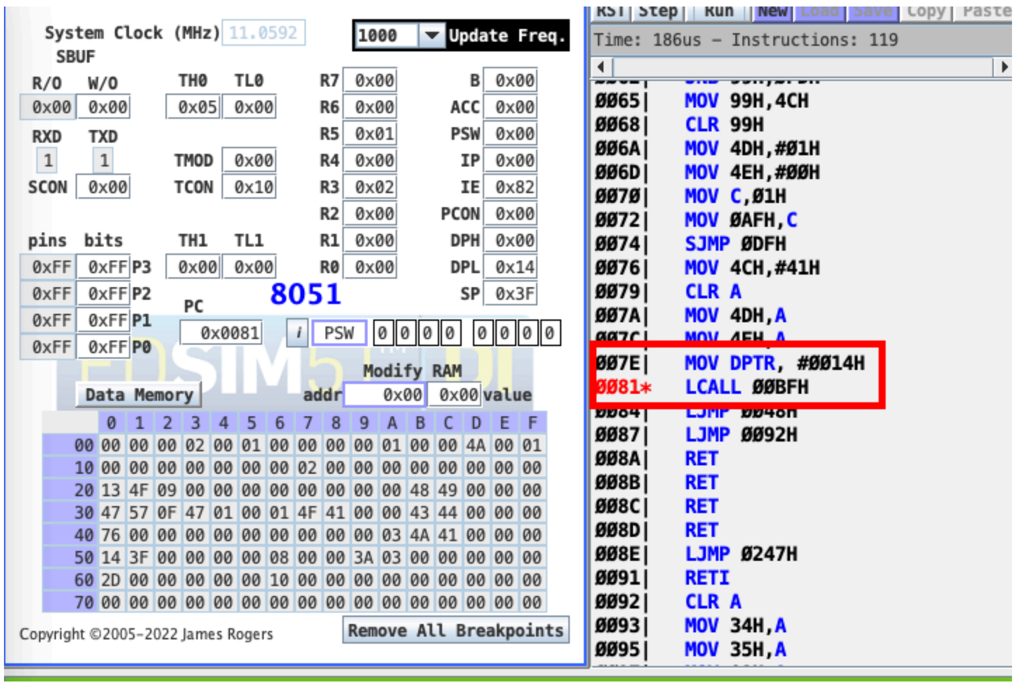
Copyright ©2005-2022 James Rogers

```
Time: 14us - Instructions: 9
ØØ7E|  MOV DPTR, #ØØ14H
ØØ81|  LCALL ØØBFH
ØØ84|  LJMP ØØ48H
ØØ87|  LJMP ØØ92H
ØØ8A|  RET
ØØ8B|  RET
ØØ8C|  RET
ØØ8D|  RET
ØØ8E|  LJMP Ø247H
ØØ91|  RETI
ØØ92|  CLR A
ØØ93|  MOV 34H,A
ØØ95|  MOV 35H,A
ØØ97|  MOV 89H,A
ØØ99|  MOV ØA8H,#82H
ØØ9C|  SETB 8CH
ØØ9E|  MOV DPTR, #ØØ76H
ØØA1*  LCALL ØØBFH
ØØA4|  MOV 35H,82H
ØØA7|  MOV A,R1
ØØA8|  PUSH ØEØH
ØØAA|  MOV A,35H
ØØAC|  ADD A,#3ØH
ØØAE|  MOV R1,A
```

Fig2.1 Screenshot before ThreadCreate(main)

When the breakpoint reached LCALL, we can see that the address of main which is 0x76 can be seen in the DPL. It will then be pushed into SP, which will change 0x07 to 0x09.

ThreadCreate(Producer);


```
System Clock (MHz) 11.0592            1000   Update Freq.
SBUF
R/O   W/O      TH0   TL0    R7 0x00      B 0x00
0x00  0x00     0x05  0x00   R6 0x00    ACC 0x00
RXD   TXD                   R5 0x01    PSW 0x00
 1     1       TMOD  0x00   R4 0x00     IP 0x00
SCON  0x00     TCON  0x10   R3 0x02     IE 0x82
                            R2 0x00   PCON 0x00
pins  bits     TH1   TL1    R1 0x00    DPH 0x00
0xFF  0xFF P3  0x00  0x00   R0 0x00    DPL 0x14
0xFF  0xFF P2           8051            SP 0x3F
0xFF  0xFF P1  PC
0xFF  0xFF P0  0x0081   i  PSW  0 0 0 0  0 0 0 0
                          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	00	00	00	02	00	01	00	00	00	00	01	00	00	4A	00	01
10	00	00	00	00	00	00	00	02	00	00	00	00	00	00	00	00
20	13	4F	09	00	00	00	00	00	00	00	00	48	49	00	00	00
30	47	57	0F	47	01	00	01	4F	41	00	00	43	44	00	00	00
40	76	00	00	00	00	00	00	00	00	00	03	4A	41	00	00	00
50	14	3F	00	00	00	00	08	00	00	3A	03	00	00	00	00	00
60	2D	00	00	00	00	00	10	00	00	00	00	00	00	00	00	00
70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Copyright ©2005-2022 James Rogers   Remove All Breakpoints

```
RST | Step | Run | New | Load | Save | Copy | Paste
Time: 186us - Instructions: 119
0065|    MOV 99H,4CH
0068|    CLR 99H
006A|    MOV 4DH,#01H
006D|    MOV 4EH,#00H
0070|    MOV C,01H
0072|    MOV 0AFH,C
0074|    SJMP 0DFH
0076|    MOV 4CH,#41H
0079|    CLR A
007A|    MOV 4DH,A
007C|    MOV 4EH,A
007E|    MOV DPTR, #0014H
0081*    LCALL 00BFH
0084|    LJMP 0048H
0087|    LJMP 0092H
008A|    RET
008B|    RET
008C|    RET
008D|    RET
008E|    LJMP 0247H
0091|    RETI
0092|    CLR A
0093|    MOV 34H,A
0095|    MOV 35H,A
```

Fig.2.2 Screenshot before ThreadCreate(Producer)

Same goes for ThreadCreate(Producer). At the breakpoint, the address of Producer which is 0x14 is pushed into the Stack Pointer, which will result in a change in the SP from 0x3F to 0x41.

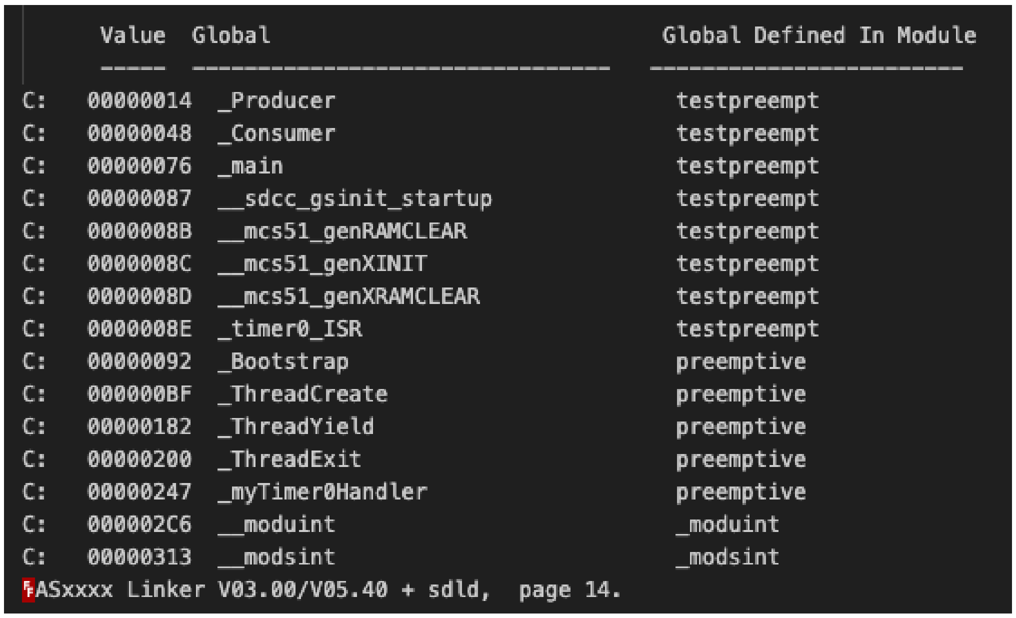
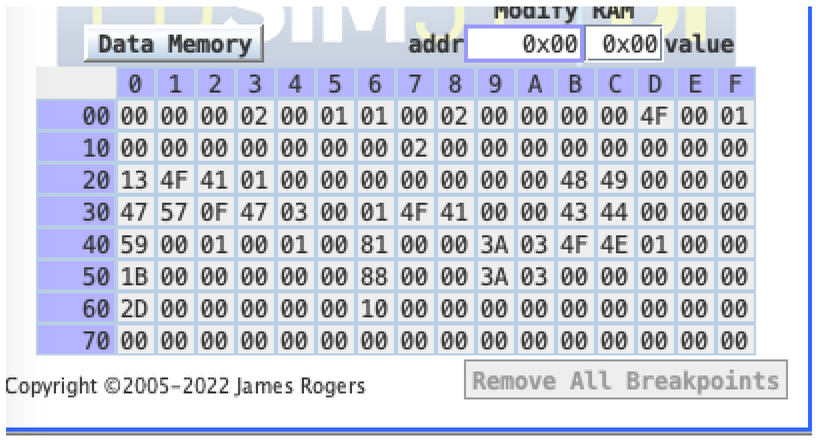

```
      Value  Global                              Global Defined In Module
      -----  --------------------------------   ------------------------
C:   00000014  _Producer                           testpreempt
C:   00000048  _Consumer                           testpreempt
C:   00000076  _main                               testpreempt
C:   00000087  __sdcc_gsinit_startup               testpreempt
C:   0000008B  __mcs51_genRAMCLEAR                 testpreempt
C:   0000008C  __mcs51_genXINIT                    testpreempt
C:   0000008D  __mcs51_genXRAMCLEAR                testpreempt
C:   0000008E  _timer0_ISR                         testpreempt
C:   00000092  _Bootstrap                          preemptive
C:   000000BF  _ThreadCreate                       preemptive
C:   00000182  _ThreadYield                        preemptive
C:   00000200  _ThreadExit                         preemptive
C:   00000247  _myTimer0Handler                    preemptive
C:   000002C6  __moduint                           _moduint
C:   00000313  __modsint                           _modsint
ASxxxx Linker V03.00/V05.40 + sdld,  page 14.
```

Fig.2.3 Function addresses value

Running Producer



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	00	00	00	02	00	01	01	00	02	00	00	00	00	4F	00	01
10	00	00	00	00	00	00	00	02	00	00	00	00	00	00	00	00
20	13	4F	41	01	00	00	00	00	00	00	00	48	49	00	00	00
30	47	57	0F	47	03	00	01	4F	41	00	00	43	44	00	00	00
40	59	00	01	00	01	00	81	00	00	3A	03	4F	4E	01	00	00
50	1B	00	00	00	00	00	88	00	00	3A	03	00	00	00	00	00
60	2D	00	00	00	00	00	10	00	00	00	00	00	00	00	00	00
70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

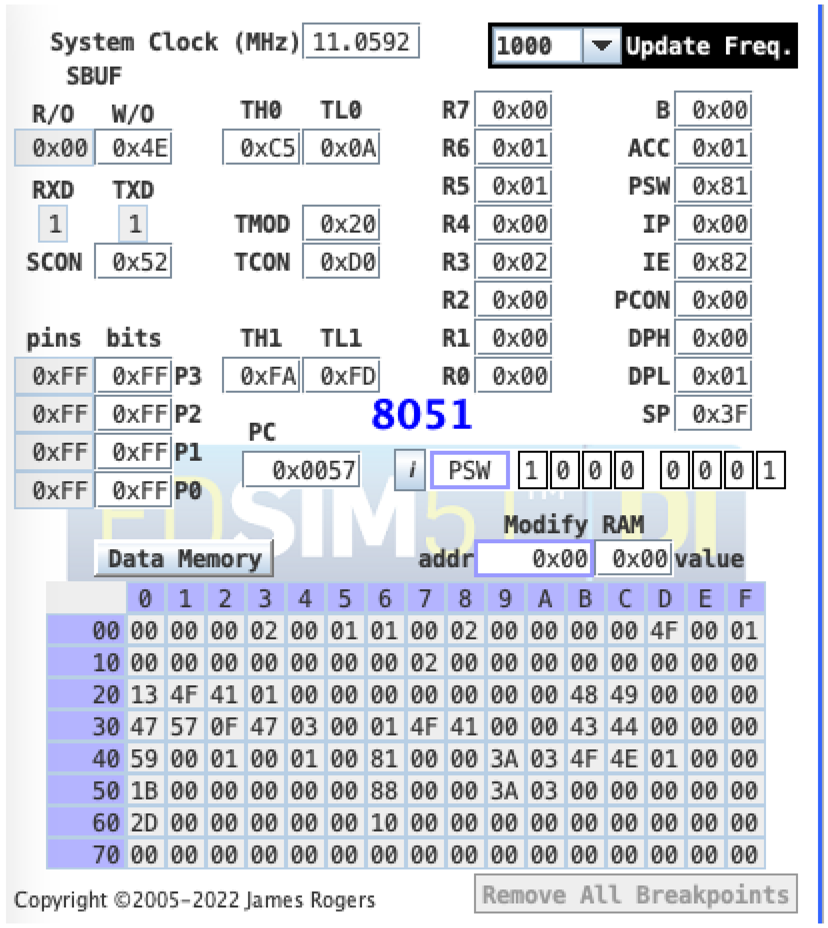
Copyright ©2005-2022 James Rogers

Remove All Breakpoints

We can see that the Producer function is running by observing the addresses of *shared_buff* and *next_buff.* In my implementation, I store *shared_buff* on 0x4C and *next_buff* on 0x4B. At each iteration, the value from 0x4B will be copied to 0x4C. At the above example, we know that the current value of the shared buffer is 4E which is HEX for the ASCII character 'N', and the next buffer has a value of 4F, which translates to 'O'.

Link of additional producer video : Drive

Running Consumer



System Clock (MHz) 11.0592

1000 Update Freq.

SBUF

R/O	W/O	TH0	TL0	R7	0x00	B	0x00
0x00	0x4E	0xC5	0x0A	R6	0x01	ACC	0x01
RXD	TXD			R5	0x01	PSW	0x81
1	1	TMOD	0x20	R4	0x00	IP	0x00
SCON	0x52	TCON	0xD0	R3	0x02	IE	0x82
				R2	0x00	PCON	0x00
pins	bits	TH1	TL1	R1	0x00	DPH	0x00
0xFF	0xFF P3	0xFA	0xFD	R0	0x00	DPL	0x01
0xFF	0xFF P2					SP	0x3F
0xFF	0xFF P1	PC					
0xFF	0xFF P0	0x0057					

8051

PSW 1 0 0 0 0 0 0 1

Modify RAM

Data Memory addr 0x00 0x00 value

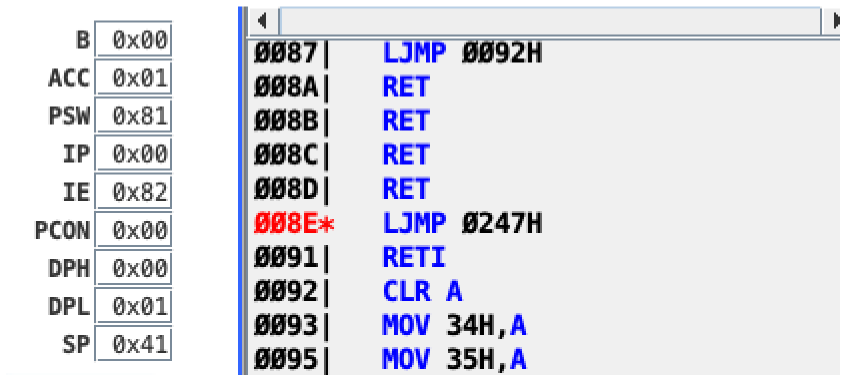
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	00	00	00	02	00	01	01	00	02	00	00	00	00	4F	00	01
10	00	00	00	00	00	00	00	02	00	00	00	00	00	00	00	00
20	13	4F	41	01	00	00	00	00	00	00	00	48	49	00	00	00
30	47	57	0F	47	03	00	01	4F	41	00	00	43	44	00	00	00
40	59	00	01	00	01	00	81	00	00	3A	03	4F	4E	01	00	00
50	1B	00	00	00	00	00	88	00	00	3A	03	00	00	00	00	00
60	2D	00	00	00	00	00	10	00	00	00	00	00	00	00	00	00
70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Copyright ©2005-2022 James Rogers

Remove All Breakpoints

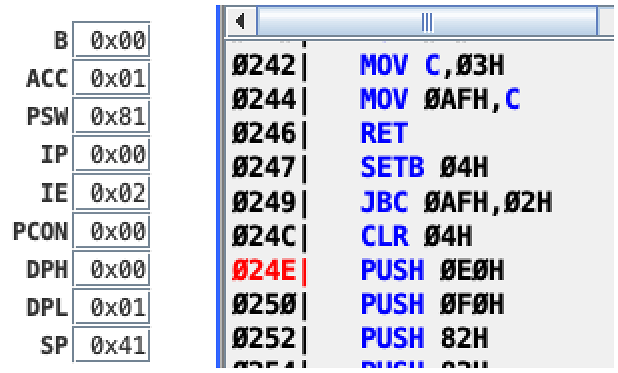
We can know Consumer is running by observing the SBUF. SBUF is currently writing out 0x4E to the received_data part on Edsim, from the shared_buffer.

Interrupt



```
   B  0x00      0087|   LJMP 0092H
 ACC  0x01      008A|   RET
 PSW  0x81      008B|   RET
  IP  0x00      008C|   RET
  IE  0x82      008D|   RET
PCON  0x00      008E*   LJMP 0247H
 DPH  0x00      0091|   RETI
 DPL  0x01      0092|   CLR A
  SP  0x41      0093|   MOV 34H,A
                0095|   MOV 35H,A
```

When the timer interrupt (0x8E) is triggered, the code jumps to 0x247, which is the address for the function myTimer0Handler(), described by the LJMP above.



```
   B  0x00      0242|   MOV C,03H
 ACC  0x01      0244|   MOV 0AFH,C
 PSW  0x81      0246|   RET
  IP  0x00      0247|   SETB 04H
  IE  0x02      0249|   JBC 0AFH,02H
PCON  0x00      024C|   CLR 04H
 DPH  0x00      024E|   PUSH 0E0H
 DPL  0x01      0250|   PUSH 0F0H
  SP  0x41      0252|   PUSH 82H
```

A few steps later, we can see that an interrupt has happened by observing the IE that changes from 0x82 to 0x02. We can tell that the interrupt is triggering on a regular basis by simply observing the changes in the IE value.