PPC 2

ID: **111061894**
Name: **Snehit**

Setup

Software	Purpose
Cygwin (3.5.4)	For Unix-like Environment
SDCC (4.4.0)	compiler suite that targets the Intel MCS51 based microprocessors
Notepad++ (8.7.1)	Write and edit .c files
EdSim51DI (2.1.36)	Simulator for 8051

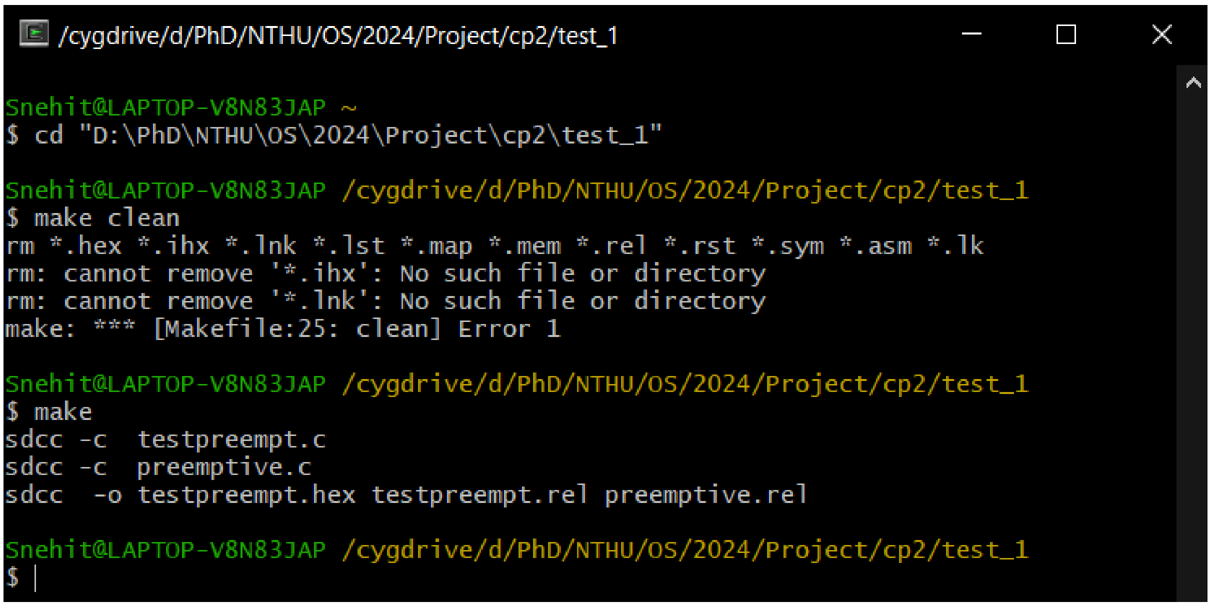
Table 1: Setup describing Software with respective version and purpose.

Creating and Compiling Makefile

Using Makefile from CP1 with some modifications. By replacing names testcoop and cooperative with testpreempt and preemptive respectively in Makefile it became compatible for CP2. Running the following commands in Cygwin (3.5.4)

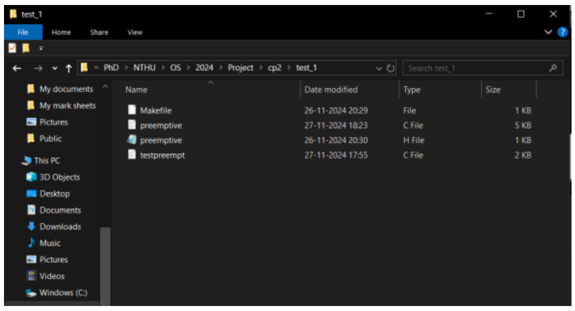
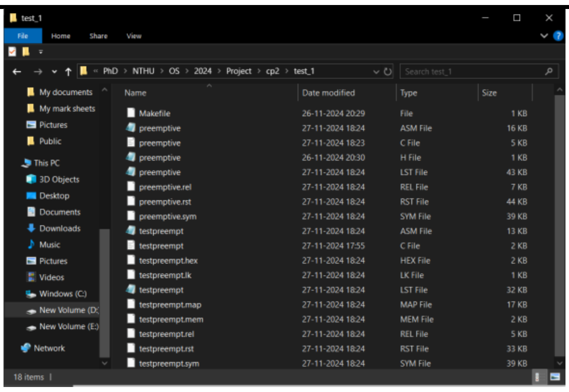
```
$ make clean
$ make
```

as shown in Fig. 1. *make clean* will clear the files generated from previous execution (if any) and then *make* command will create new require file as per the code written in *.c* files. Table 2 shows the result of respective make command.



```
/cygdrive/d/PhD/NTHU/OS/2024/Project/cp2/test_1

Snehit@LAPTOP-V8N83JAP ~
$ cd "D:\PhD\NTHU\OS\2024\Project\cp2\test_1"

Snehit@LAPTOP-V8N83JAP /cygdrive/d/PhD/NTHU/OS/2024/Project/cp2/test_1
$ make clean
rm *.hex *.ihx *.lnk *.lst *.map *.mem *.rel *.rst *.sym *.asm *.lk
rm: cannot remove '*.ihx': No such file or directory
rm: cannot remove '*.lnk': No such file or directory
make: *** [Makefile:25: clean] Error 1

Snehit@LAPTOP-V8N83JAP /cygdrive/d/PhD/NTHU/OS/2024/Project/cp2/test_1
$ make
sdcc -c  testpreempt.c
sdcc -c  preemptive.c
sdcc  -o testpreempt.hex testpreempt.rel preemptive.rel

Snehit@LAPTOP-V8N83JAP /cygdrive/d/PhD/NTHU/OS/2024/Project/cp2/test_1
$ |
```

Fig. 1: Screenshot of Cygwin after running *make clean* and *make* command.

After $ *make clean*	**After $ *make***
	

After $ make clean:

```
test_1
PhD > NTHU > OS > 2024 > Project > cp2 > test_1

Name              Date modified      Type      Size
Makefile          26-11-2024 20:29   File      1 KB
preemptive        27-11-2024 18:23   C File    5 KB
preemptive        26-11-2024 20:30   H File    1 KB
testpreempt       27-11-2024 17:55   C File    2 KB
```

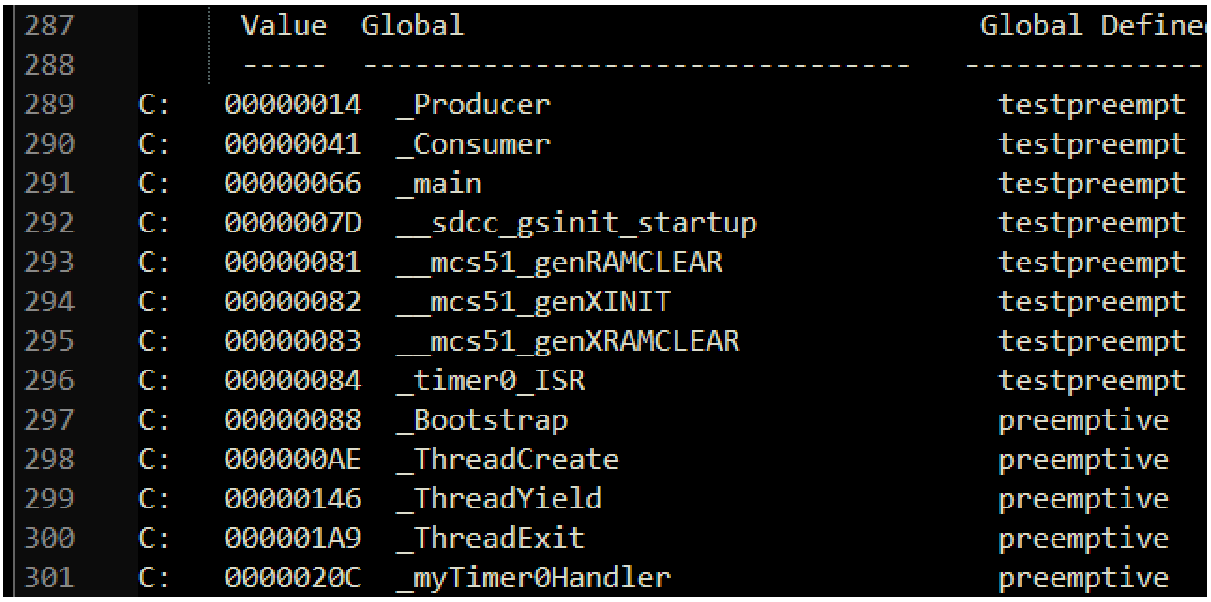
After $ make:

```
test_1
PhD > NTHU > OS > 2024 > Project > cp2 > test_1

Name              Date modified      Type       Size
Makefile          26-11-2024 20:29   File       1 KB
preemptive        27-11-2024 18:24   ASM File   16 KB
preemptive        27-11-2024 18:23   C File     5 KB
preemptive        26-11-2024 20:30   H File     1 KB
preemptive        27-11-2024 18:24   LST File   43 KB
preemptive.rel    27-11-2024 18:24   REL File   7 KB
preemptive.rst    27-11-2024 18:24   RST File   44 KB
preemptive.sym    27-11-2024 18:24   SYM File   39 KB
testpreempt       27-11-2024 18:24   ASM File   13 KB
testpreempt       27-11-2024 17:55   C File     2 KB
testpreempt.hex   27-11-2024 18:24   HEX File   2 KB
testpreempt.lk    27-11-2024 18:24   LK File    1 KB
testpreempt       27-11-2024 18:24   LST File   32 KB
testpreempt.map   27-11-2024 18:24   MAP File   17 KB
testpreempt.mem   27-11-2024 18:24   MEM File   2 KB
testpreempt.rel   27-11-2024 18:24   REL File   5 KB
testpreempt.rst   27-11-2024 18:24   RST File   33 KB
testpreempt.sym   27-11-2024 18:24   SYM File   39 KB
18 items
```

Table 2: results of Makefile compilation

ThreadCreate calls

There are two thread create calls one for main and another for Producer.

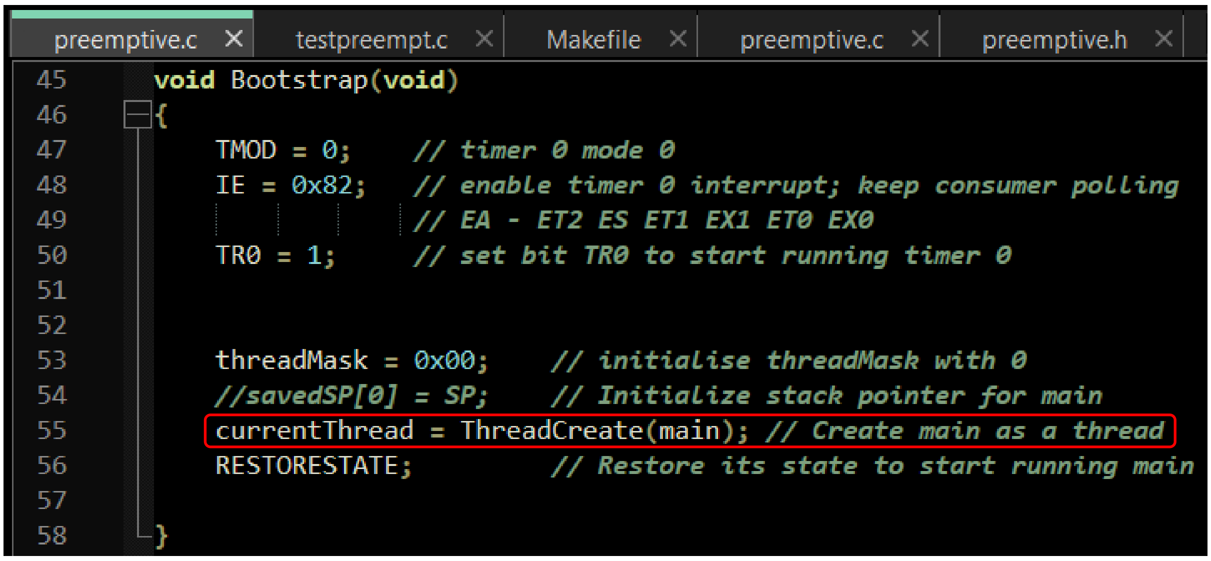


```
        Value   Global                              Global Define
        -----   --------------------------------    --------------
  C:   00000014  _Producer                           testpreempt
  C:   00000041  _Consumer                           testpreempt
  C:   00000066  _main                               testpreempt
  C:   0000007D  __sdcc_gsinit_startup               testpreempt
  C:   00000081  __mcs51_genRAMCLEAR                 testpreempt
  C:   00000082  __mcs51_genXINIT                    testpreempt
  C:   00000083  __mcs51_genXRAMCLEAR                testpreempt
  C:   00000084  _timer0_ISR                         testpreempt
  C:   00000088  _Bootstrap                          preemptive
  C:   000000AE  _ThreadCreate                       preemptive
  C:   00000146  _ThreadYield                        preemptive
  C:   000001A9  _ThreadExit                         preemptive
  C:   0000020C  _myTimer0Handler                    preemptive
```

Fig 2: Address of respective functions in preemptive and testpreempt for reference.

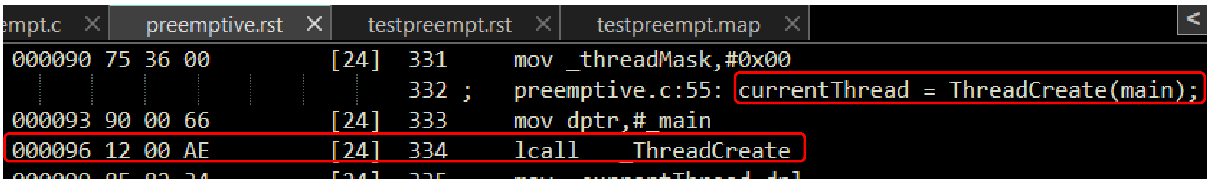
1. ***ThreadCreate(main)***

ThreadCreate for main form preemptive is called in during startup using Boostrap as is shown in code snippet in Fig. 3.



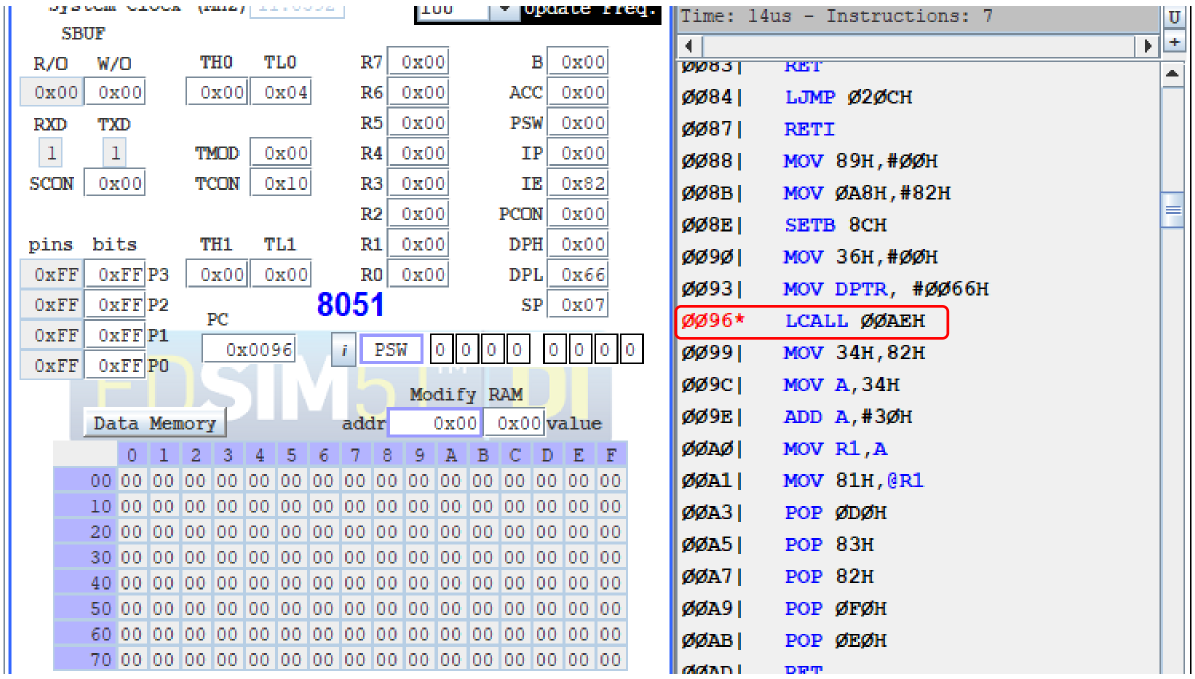
```
void Bootstrap(void)
{
    TMOD = 0;     // timer 0 mode 0
    IE = 0x82;    // enable timer 0 interrupt; keep consumer polling
                  // EA - ET2 ES ET1 EX1 ET0 EX0
    TR0 = 1;      // set bit TR0 to start running timer 0


    threadMask = 0x00;     // initialise threadMask with 0
    //savedSP[0] = SP;     // Initialize stack pointer for main
    currentThread = ThreadCreate(main); // Create main as a thread
    RESTORESTATE;          // Restore its state to start running main

}
```

Fig 3: ThreadCreate(main) in Boostrap



```
000090 75 36 00          [24]  331      mov _threadMask,#0x00
                               332 ;    preemptive.c:55: currentThread = ThreadCreate(main);
000093 90 00 66          [24]  333      mov dptr,#_main
000096 12 00 AE          [24]  334      lcall   _ThreadCreate
```

Fig. 4: ThreadCreate(main) call indicating at address 0096H.



```
SBUF
R/O   W/O       TH0   TL0      R7 0x00      B 0x00
0x00  0x00      0x00  0x04     R6 0x00    ACC 0x00
RXD   TXD                      R5 0x00    PSW 0x00
 1     1        TMOD  0x00     R4 0x00     IP 0x00
SCON  0x00      TCON  0x10     R3 0x00     IE 0x82
                               R2 0x00   PCON 0x00
pins  bits      TH1   TL1      R1 0x00    DPH 0x00
0xFF  0xFF P3   0x00  0x00     R0 0x00    DPL 0x66
0xFF  0xFF P2              8051            SP 0x07
0xFF  0xFF P1   PC
0xFF  0xFF P0   0x0096   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 00 00 00 00 00 00 00 00 00 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
50 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
60 00 00 00 00 00 00 00 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
```

```
Time: 14us - Instructions: 7
0083|   RET
0084|   LJMP 020CH
0087|   RETI
0088|   MOV 89H,#00H
008B|   MOV 0A8H,#82H
008E|   SETB 8CH
0090|   MOV 36H,#00H
0093|   MOV DPTR, #0066H
0096*   LCALL 00AEH
0099|   MOV 34H,82H
009C|   MOV A,34H
009E|   ADD A,#30H
00A0|   MOV R1,A
00A1|   MOV 81H,@R1
00A3|   POP 0D0H
00A5|   POP 83H
00A7|   POP 82H
00A9|   POP 0F0H
00AB|   POP 0E0H
00AD|   RET
```

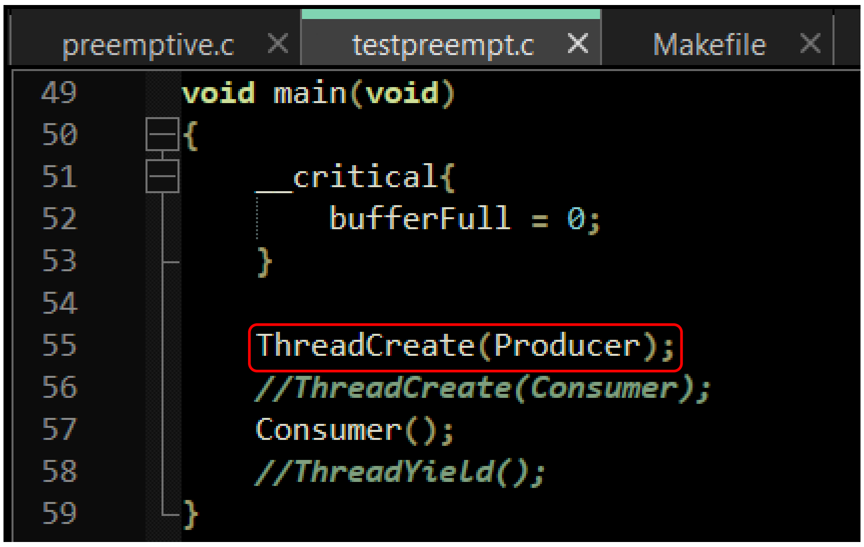
Fig. 5: Screenshot EdSim51 BreakPoint at 0096H

As can be seen from Fig. 4 ThreadCreate(main) is called at address 0096H, hence BreakPoint on 0096H in EdSim51 is added which perform "LCALL 00AEH" refer Figure 5.

DPTR (i.e. DPH and DPL) is loaded with 0066H which is address of main in testpreempt (refer Figure 2) and SP is at 07H which is default value at SP depicting nothing is loaded into SP yet.

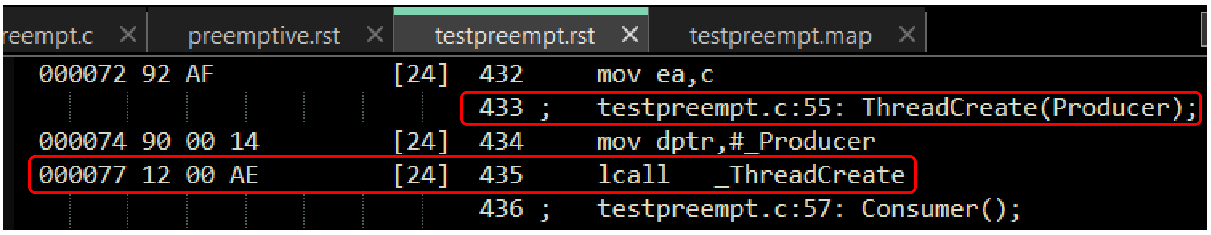
2. ThreadCreate(Producer)

ThreadCreate for Producer is called in main of testpreempt (refer Fig. 6), where Producer maps to 0014H (refer Figure 2) and ThreadCreate call for Producer is at 0077H can be observed in Fig. 7.



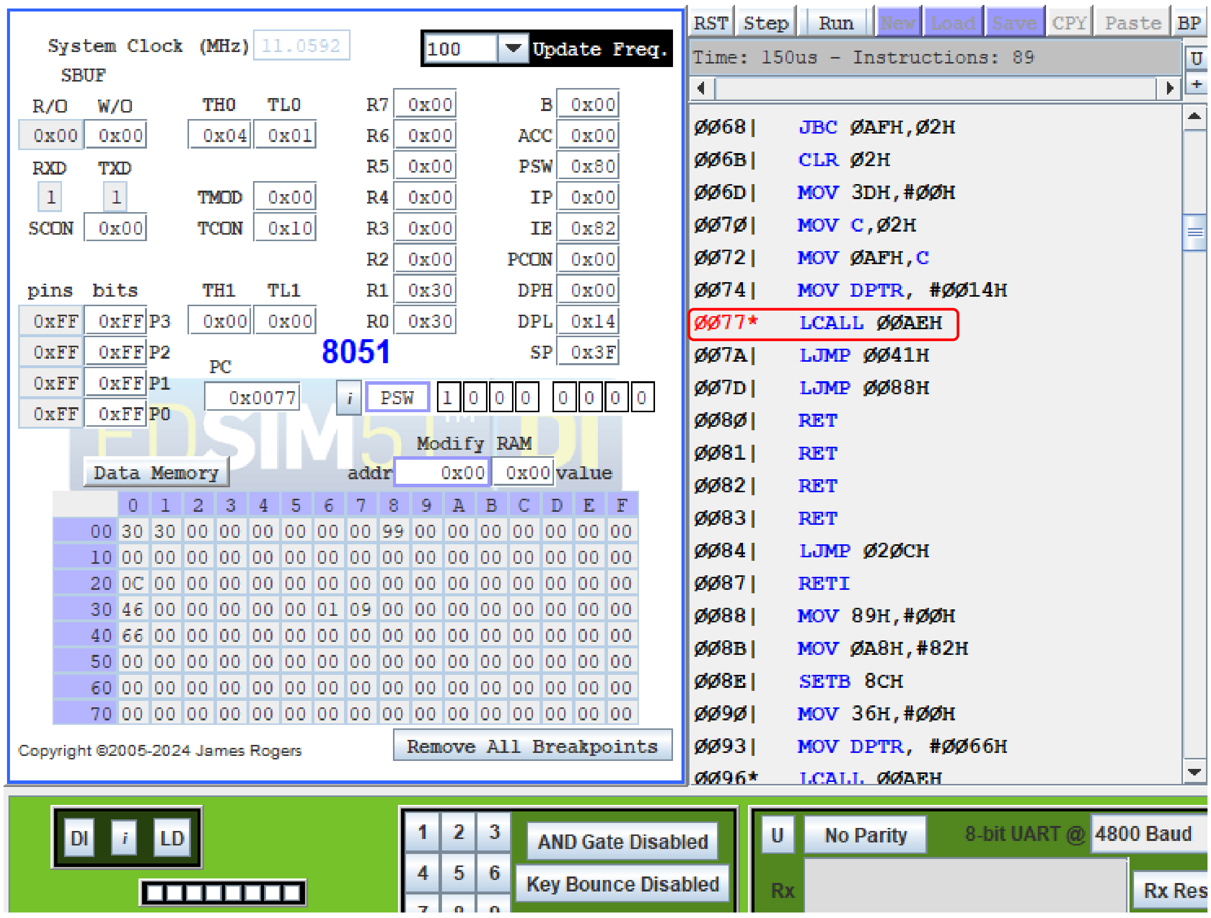
```c
void main(void)
{
    __critical{
        bufferFull = 0;
    }

    ThreadCreate(Producer);
    //ThreadCreate(Consumer);
    Consumer();
    //ThreadYield();
}
```

Fig 6: ThreadCreate(main) in Boostrap



```
000072 92 AF          [24]  432     mov ea,c
                            433 ;   testpreempt.c:55: ThreadCreate(Producer);
000074 90 00 14       [24]  434     mov dptr,#_Producer
000077 12 00 AE       [24]  435     lcall   _ThreadCreate
                            436 ;   testpreempt.c:57: Consumer();
```

Fig. 7: ThreadCreate(Producer) call indicating at address 0077H.



```
System Clock (MHz) 11.0592        100
SBUF
R/O   W/O     TH0   TL0     R7 0x00     B    0x00
0x00  0x00    0x04  0x01    R6 0x00     ACC  0x00
RXD   TXD                   R5 0x00     PSW  0x80
1     1       TMOD  0x00    R4 0x00     IP   0x00
SCON  0x00    TCON  0x10    R3 0x00     IE   0x82
                            R2 0x00     PCON 0x00
pins  bits    TH1   TL1     R1 0x30     DPH  0x00
0xFF  0xFF P3 0x00  0x00    R0 0x30     DPL  0x14
0xFF  0xFF P2                           SP   0x3F
0xFF  0xFF P1   PC
0xFF  0xFF P0   0x0077   PSW 1 0 0 0 0 0 0 0
addr 0x00  0x00 value
```

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	30	30	00	00	00	00	00	00	99	00	00	00	00	00	00	00
10	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
20	0C	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
30	46	00	00	00	00	00	01	09	00	00	00	00	00	00	00	00
40	66	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
50	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
60	00	00	00	00	00	00	00	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 ©2005-2024 James Rogers

```
Time: 150us - Instructions: 89

0068|   JBC ØAFH,Ø2H
006B|   CLR Ø2H
006D|   MOV 3DH,#ØØH
0070|   MOV C,Ø2H
0072|   MOV ØAFH,C
0074|   MOV DPTR, #ØØ14H
0077*   LCALL ØØAEH
007A|   LJMP ØØ41H
007D|   LJMP ØØ88H
0080|   RET
0081|   RET
0082|   RET
0083|   RET
0084|   LJMP Ø2ØCH
0087|   RETI
0088|   MOV 89H,#ØØH
008B|   MOV ØA8H,#82H
008E|   SETB 8CH
0090|   MOV 36H,#ØØH
0093|   MOV DPTR, #ØØ66H
0096*   LCALL ØØAEH
```

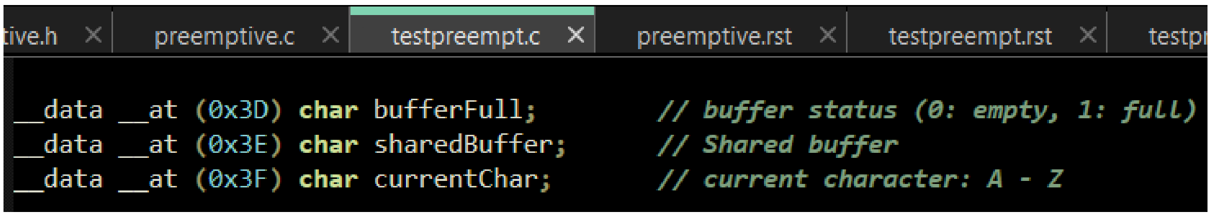
```
1 2 3    AND Gate Disabled
4 5 6    Key Bounce Disabled
8-bit UART @ 4800 Baud
```

Fig. 8: Screenshot EdSim51 BreakPoint at 0077H

DPTR (i.e. DPH and DPL) having 0014H which is Producer's address (refer Figure 2), and SP is at 3FH (refer Fig. 8). On the BreakPoint at 0077H it's going to call for 00AEH which is address of ThreadCreate (refer Figure 2).

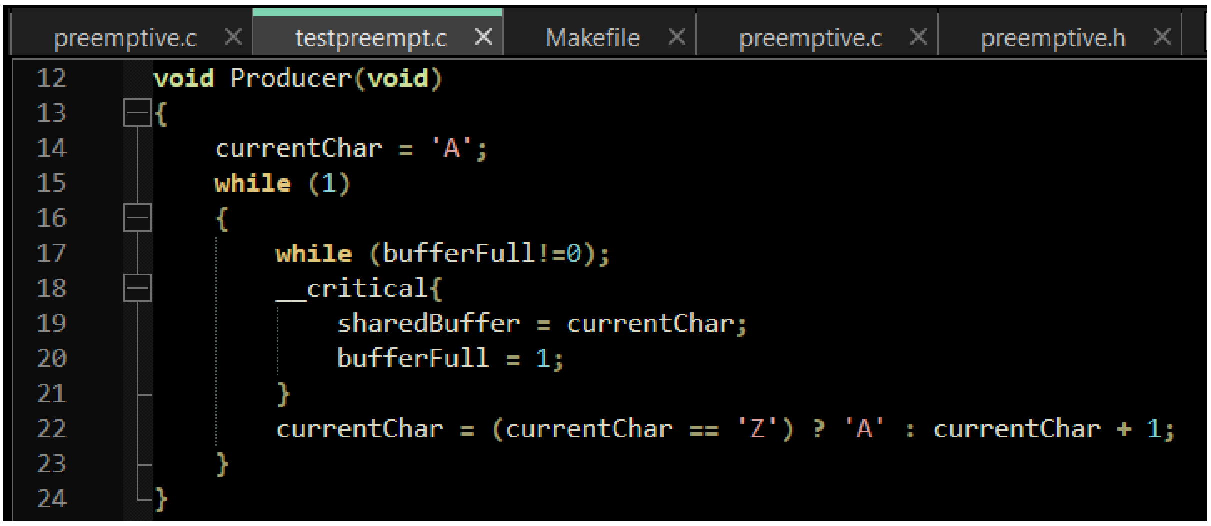
Producer in Run

In Producer, variable "currentChar" is used to loop through the character "A" to "Z" and repeat the cycle again and it's going to be assigned to variable "sharedBuffer" to later which will be then transferred to SBUF in EdSim51 (refer Fig. 10 for code snippet). Variable "bufferFull" is a common variable for Producer and consumer to update, during producer is running if sharedBuffer contains new character "bufferFull" is triggered to 1 and when it comes to consumer once character transferred SBUF "bufferFull" is set to 0. During preemption, "bufferFull" and "sharedBuffer" is wrapped in __critical { }, since those are the common variable used by producer and consumer.



```c
__data __at (0x3D) char bufferFull;        // buffer status (0: empty, 1: full)
__data __at (0x3E) char sharedBuffer;      // Shared buffer
__data __at (0x3F) char currentChar;       // current character: A - Z
```

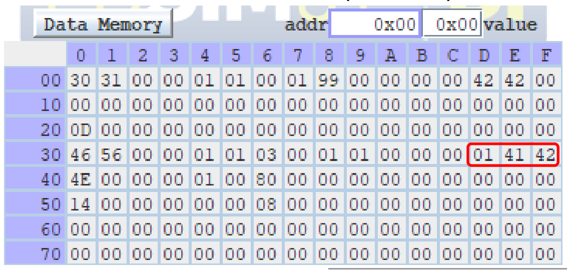
Fig. 9: Screenshot testpreempt.c indicating address for respective variables



```c
void Producer(void)
{
    currentChar = 'A';
    while (1)
    {
        while (bufferFull!=0);
        __critical{
            sharedBuffer = currentChar;
            bufferFull = 1;
        }
        currentChar = (currentChar == 'Z') ? 'A' : currentChar + 1;
    }
}
```

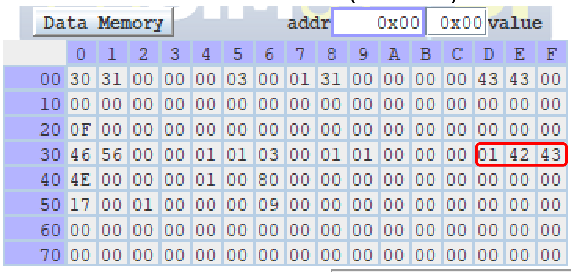
Fig 10: Producer code snippet

sharedBuffer = "A" (i.e. 41H)
currentChar = "B" (i.e. 42H)



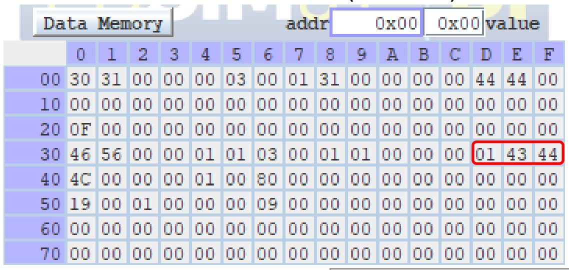
```
                      addr     0x00  0x00 value
    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
00 30 31 00 00 01 01 00 01 99 00 00 00 00 42 42 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
20 0D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
30 46 56 00 00 01 01 03 00 01 01 00 00 00 01 41 42
40 4E 00 00 00 01 00 80 00 00 00 00 00 00 00 00 00
50 14 00 00 00 00 00 08 00 00 00 00 00 00 00 00 00
60 00 00 00 00 00 00 00 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
```

sharedBuffer = "B" (i.e. 42H)
currentChar = "C" (i.e. 43H)



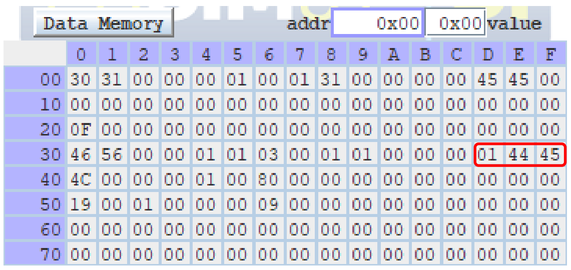
```
                      addr     0x00  0x00 value
    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
00 30 31 00 00 00 03 00 01 31 00 00 00 00 43 43 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
20 0F 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
30 46 56 00 00 01 01 03 00 01 01 00 00 00 01 42 43
40 4E 00 00 00 01 00 80 00 00 00 00 00 00 00 00 00
50 17 00 01 00 00 00 09 00 00 00 00 00 00 00 00 00
60 00 00 00 00 00 00 00 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
```

sharedBuffer = "C" (i.e. 43H)
currentChar = "D" (i.e. 44H)



```
                      addr     0x00  0x00 value
    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
00 30 31 00 00 00 03 00 01 31 00 00 00 00 44 44 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
20 0F 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
30 46 56 00 00 01 01 03 00 01 01 00 00 00 01 43 44
40 4C 00 00 00 01 00 80 00 00 00 00 00 00 00 00 00
50 19 00 01 00 00 00 09 00 00 00 00 00 00 00 00 00
60 00 00 00 00 00 00 00 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
```

sharedBuffer = "D" (i.e. 44H)
currentChar = "E" (i.e. 45H)



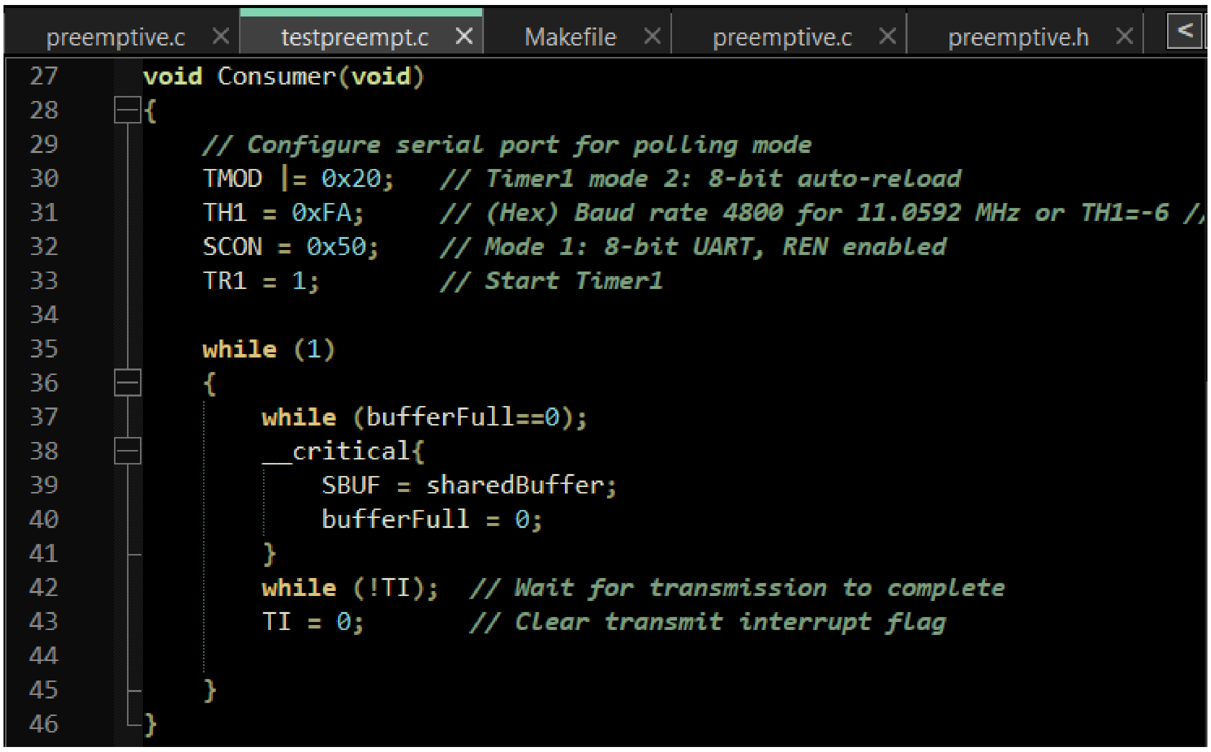
```
                      addr     0x00  0x00 value
    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
00 30 31 00 00 00 01 00 01 31 00 00 00 00 45 45 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
20 0F 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
30 46 56 00 00 01 01 03 00 01 01 00 00 00 01 44 45
40 4C 00 00 00 01 00 80 00 00 00 00 00 00 00 00 00
50 19 00 01 00 00 00 09 00 00 00 00 00 00 00 00 00
60 00 00 00 00 00 00 00 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
```

Table 3: Status of variables at respective memory during Producer is running.

Observing 3EH and 3FH as from Figure 9 indicating address of "sharedBuffer" and "currentChar" we can see in Table 3 that 3EH gets updated with value from 3FH and which is hex values of character "A" to "Z". Noting that this update code is made inside producer, and observing this cycle is executing shown in table indicates that producer is running. Also observe the value at 3DH which is 1H indicates buffer status and it is set to 1H once producer produce and in table it is clear to observe that it is set to 1H means producer is running.

Consumer in Run

When Consumer is running the SBUF receives the character from “sharedBuffer” and can be displayed on UART Receiver results are as shown in Figure 12 (a) and 12 (b) whereas refer Fig 11 code snippet for consumer.



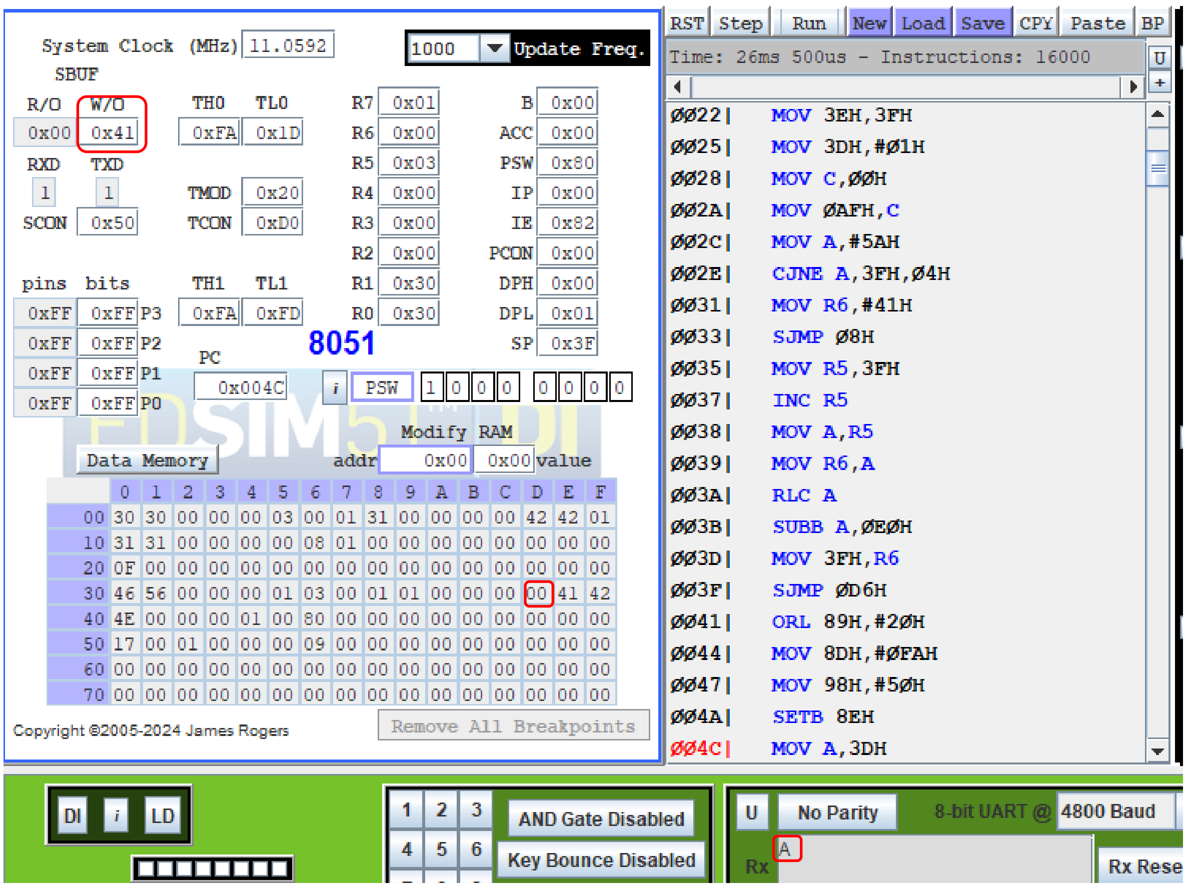
preemptive.c testpreempt.c Makefile preemptive.c preemptive.h

```
void Consumer(void)
{
    // Configure serial port for polling mode
    TMOD |= 0x20;    // Timer1 mode 2: 8-bit auto-reload
    TH1 = 0xFA;      // (Hex) Baud rate 4800 for 11.0592 MHz or TH1=-6 /
    SCON = 0x50;     // Mode 1: 8-bit UART, REN enabled
    TR1 = 1;         // Start Timer1

    while (1)
    {
        while (bufferFull==0);
        __critical{
            SBUF = sharedBuffer;
            bufferFull = 0;
        }
        while (!TI);  // Wait for transmission to complete
        TI = 0;       // Clear transmit interrupt flag

    }
}
```

Fig 11: Producer code snippet

In Figure 12 (a), Consumer running with SBUF having W/O 0x41H (i.e. “A”) which is the character to displayed in UART receiver. And in Figure 12 (b), Consumer running with SBUF having W/O 0x42H (i.e. “B”) which will be the next character to be displayed in UART receiver along with previous character (i.e. “AB”).

Along with that we can observe “bufferFull” at address 0x3DH (refer Fig. 9) is set to 0 during consumer is running. Figure 12 (a & b) shows 0x3DH is set to 0H which means consumer is in running.



```
System Clock (MHz) 11.0592        1000  Update Freq.
SBUF
R/O   W/O      TH0   TL0     R7 0x01     B    0x00
0x00  0x41     0xFA  0x1D    R6 0x00     ACC  0x00
RXD   TXD                    R5 0x03     PSW  0x80
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 0x30     DPH  0x00
0xFF  0xFF P3  0xFA  0xFD    R0 0x30     DPL  0x01
0xFF  0xFF P2              8051          SP   0x3F
0xFF  0xFF P1      PC
0xFF  0xFF P0      0x004C    PSW 1 0 0 0 0 0 0 0
                                Modify RAM
Data Memory               addr 0x00  0x00 value
```

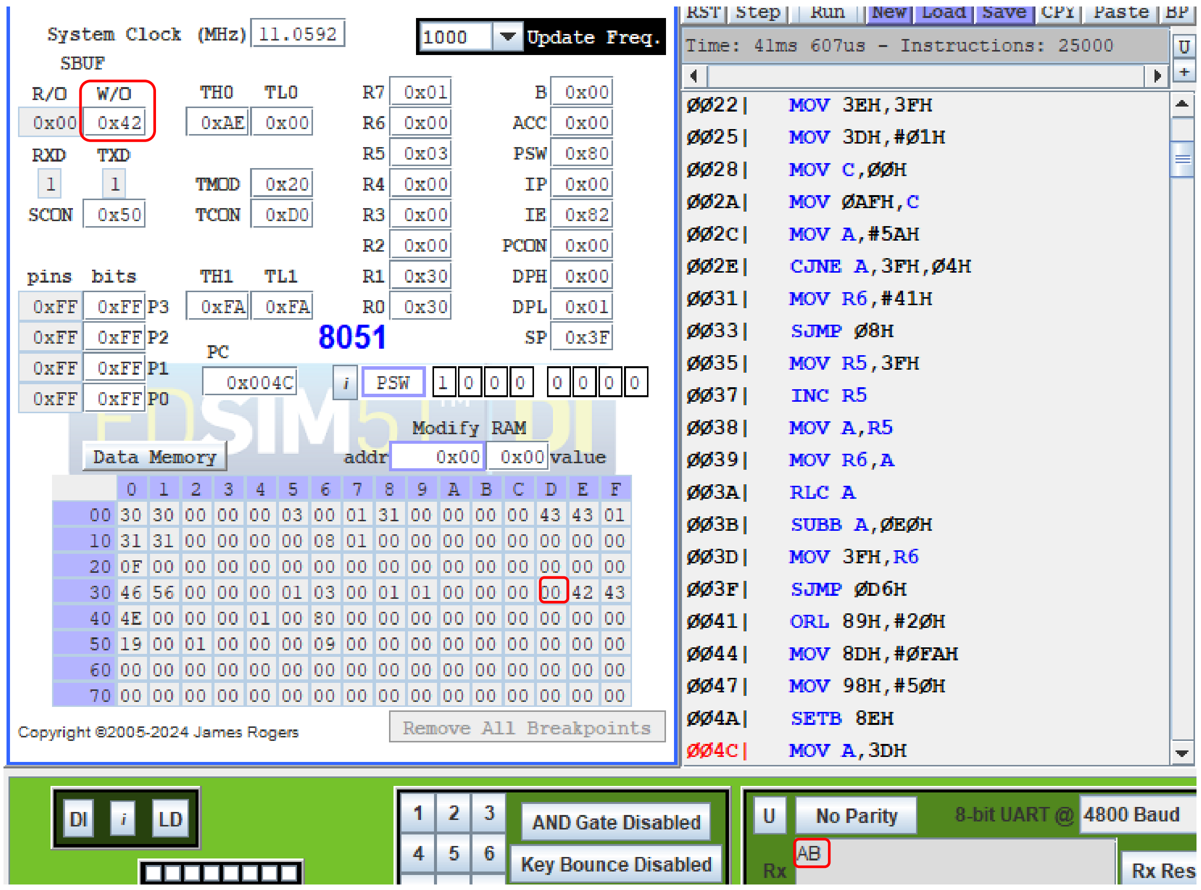
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	30	30	00	00	00	03	00	01	31	00	00	00	00	42	42	01
10	31	31	00	00	00	00	08	01	00	00	00	00	00	00	00	00
20	0F	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
30	46	56	00	00	00	01	03	00	01	01	00	00	00	00	41	42
40	4E	00	00	00	01	00	80	00	00	00	00	00	00	00	00	00
50	17	00	01	00	00	00	09	00	00	00	00	00	00	00	00	00
60	00	00	00	00	00	00	00	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 ©2005-2024 James Rogers   Remove All Breakpoints

```
RST | Step | Run | New | Load | Save | CPY | Paste | BP
Time: 26ms 500us - Instructions: 16000
0022| MOV 3EH,3FH
0025| MOV 3DH,#01H
0028| MOV C,00H
002A| MOV 0AFH,C
002C| MOV A,#5AH
002E| CJNE A,3FH,04H
0031| MOV R6,#41H
0033| SJMP 08H
0035| MOV R5,3FH
0037| INC R5
0038| MOV A,R5
0039| MOV R6,A
003A| RLC A
003B| SUBB A,0E0H
003D| MOV 3FH,R6
003F| SJMP 0D6H
0041| ORL 89H,#20H
0044| MOV 8DH,#0FAH
0047| MOV 98H,#50H
004A| SETB 8EH
004C| MOV A,3DH
```

```
DI  i  LD     1 2 3  AND Gate Disabled     U  No Parity  8-bit UART @ 4800 Baud
              4 5 6  Key Bounce Disabled   Rx A                      Rx Rese
```

Fig. 12 (a): Screenshot (1) of EdSim51 while Consumer is running.



```
System Clock (MHz) 11.0592        1000  Update Freq.
SBUF
R/O   W/O      TH0   TL0     R7 0x01     B    0x00
0x00  0x42     0xAE  0x00    R6 0x00     ACC  0x00
RXD   TXD                    R5 0x03     PSW  0x80
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 0x30     DPH  0x00
0xFF  0xFF P3  0xFA  0xFA    R0 0x30     DPL  0x01
0xFF  0xFF P2              8051          SP   0x3F
0xFF  0xFF P1      PC
0xFF  0xFF P0      0x004C    PSW 1 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	30	30	00	00	00	03	00	01	31	00	00	00	00	43	43	01
10	31	31	00	00	00	00	08	01	00	00	00	00	00	00	00	00
20	0F	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
30	46	56	00	00	00	01	03	00	01	01	00	00	00	00	42	43
40	4E	00	00	00	01	00	80	00	00	00	00	00	00	00	00	00
50	19	00	01	00	00	00	09	00	00	00	00	00	00	00	00	00
60	00	00	00	00	00	00	00	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 ©2005-2024 James Rogers   Remove All Breakpoints

```
RST | Step | Run | New | Load | Save | CPY | Paste | BP
Time: 41ms 607us - Instructions: 25000
0022| MOV 3EH,3FH
0025| MOV 3DH,#01H
0028| MOV C,00H
002A| MOV 0AFH,C
002C| MOV A,#5AH
002E| CJNE A,3FH,04H
0031| MOV R6,#41H
0033| SJMP 08H
0035| MOV R5,3FH
0037| INC R5
0038| MOV A,R5
0039| MOV R6,A
003A| RLC A
003B| SUBB A,0E0H
003D| MOV 3FH,R6
003F| SJMP 0D6H
0041| ORL 89H,#20H
0044| MOV 8DH,#0FAH
0047| MOV 98H,#50H
004A| SETB 8EH
004C| MOV A,3DH
```

```
DI  i  LD     1 2 3  AND Gate Disabled     U  No Parity  8-bit UART @ 4800 Baud
              4 5 6  Key Bounce Disabled   Rx AB                     Rx Res
```

Fig. 12 (b): Screenshot (2) of EdSim51 while Consumer is running.

Interrupt triggering on a regular basis

Timer 0 in mode 0 is 13 bit timer where 8 bits from TH0 and 5 bits from TL0 are used in cycle to count from 0000H to 1FFFH (i.e. 8192 cycle) once reached 1FFFH it will reset to 0000H and the cycle repeats.

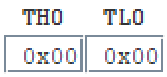
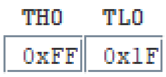
Timer 0 (mode 0) at 0H (i.e. start)	Timer 0 (mode 0) at 1FFFH (i.e. end)
 TH0 TL0 0x00 0x00	 TH0 TL0 0xFF 0x1F

Table 4: Screenshot of Timer at start and end of cycle

As suggested in lecture slide “08-timer-preemption”, “myTimer0Handler” start with EA=0 and end with EA=1 that means during “timer0_ISR” bit 7 of Interrupt Enable is off and turn on after.

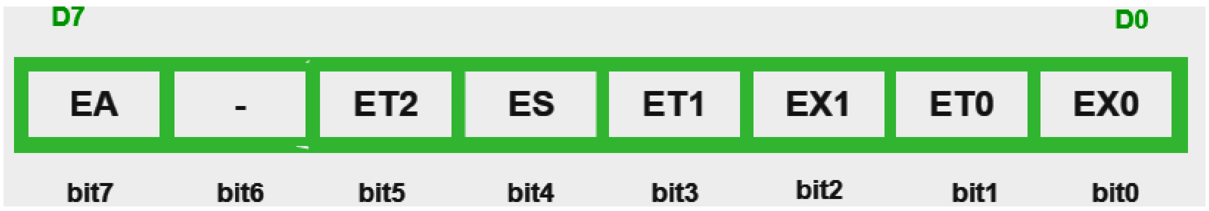
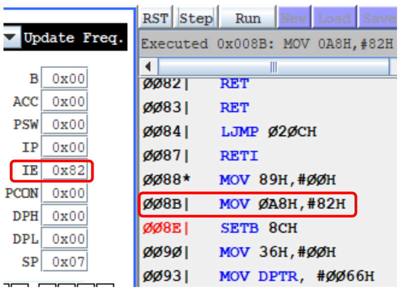
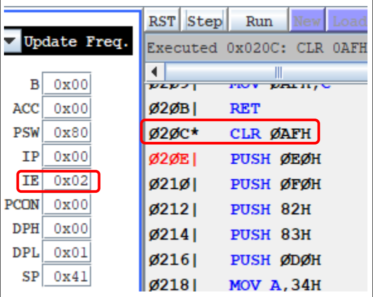
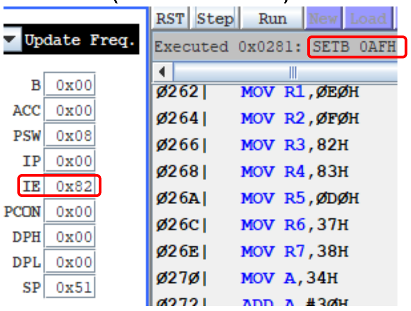


Fig 13: IE (Interrupt Enable) Register (Source geeksforgeeks.org)

This can be observed in EdSim51 at IE, EA=0 correspond to IE=0x02H, EA=1 to IE=0x82H.

IE=0x82H (Bootstrap)	EA=0 (myTimer0Handler) (i.e. IE=0x02H)	EA=1 (myTimer0Handler) (i.e. IE=0x82H)
		

```
Executed 0x008B: MOV 0A8H,#82H
B     0x00    ØØ82|   RET
ACC   0x00    ØØ83|   RET
PSW   0x00    ØØ84|   LJMP Ø2ØCH
IP    0x00    ØØ87|   RETI
IE    0x82    ØØ88*   MOV 89H,#ØØH
PCON  0x00    ØØ8B|   MOV ØA8H,#82H
DPH   0x00    ØØ8E|   SETB 8CH
DPL   0x00    ØØ9Ø|   MOV 36H,#ØØH
SP    0x07    ØØ93|   MOV DPTR, #ØØ66H
```

```
Executed 0x020C: CLR 0AFH
B     0x00    Ø2ØB|   RET
ACC   0x00    Ø2ØC*   CLR ØAFH
PSW   0x80    Ø2ØE|   PUSH ØEØH
IP    0x00    Ø21Ø|   PUSH ØFØH
IE    0x02    Ø212|   PUSH 82H
PCON  0x00    Ø214|   PUSH 83H
DPH   0x00    Ø216|   PUSH ØDØH
DPL   0x01    Ø218|   MOV A,34H
SP    0x41
```

```
Executed 0x0281: SETB 0AFH
B     0x00    Ø262|   MOV R1,ØEØH
ACC   0x00    Ø264|   MOV R2,ØFØH
PSW   0x08    Ø266|   MOV R3,82H
IP    0x00    Ø268|   MOV R4,83H
IE    0x82    Ø26A|   MOV R5,ØDØH
PCON  0x00    Ø26C|   MOV R6,37H
DPH   0x00    Ø26E|   MOV R7,38H
DPL   0x00    Ø27Ø|   MOV A,34H
SP    0x51
```

Table 5: Interrupt triggering (IE)

Onward during execution IE keeps switching between 0x82H and 0x02H in regular intervals, which concludes that the interrupt is triggering on regular intervals.