PPC 1

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.

Compiling Makefile

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.

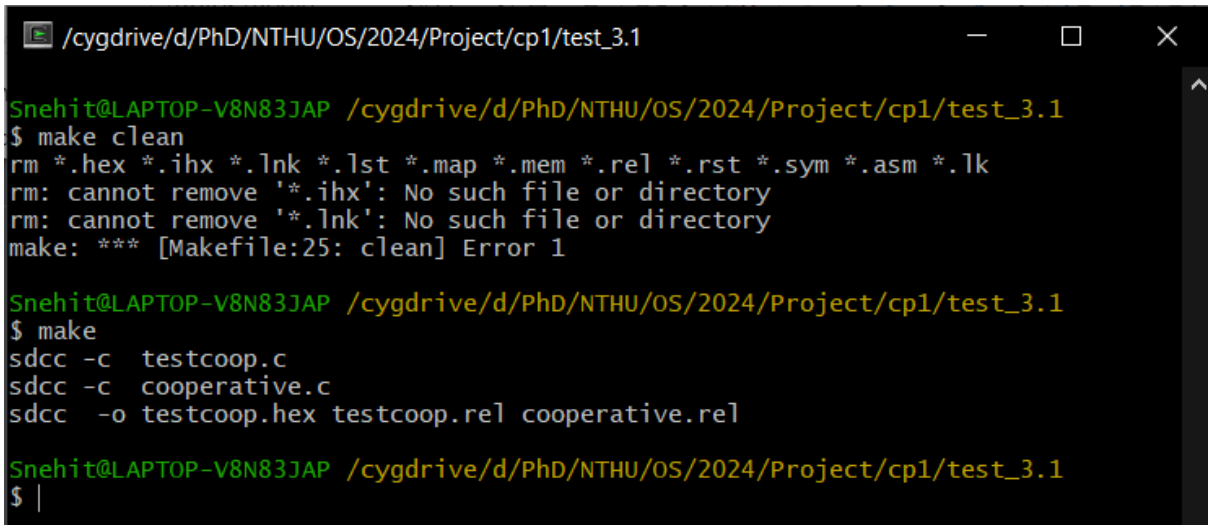


Fig. 1: Screenshot of Cygwin after running *make clean* and *make* command.

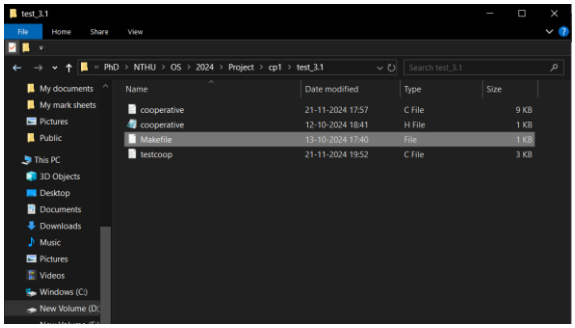
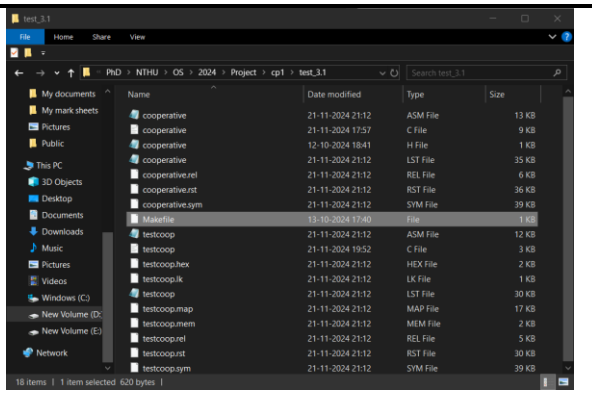
After $ *make clean*	**After $ *make***
	

Table 2: results of Makefile compilation

ThreadCreate calls

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

```
oop.c   cooperative.c   testcoop.map   testcoop.rst   cooper

      Value   Global                                  Global Define
      -----   --------------------------------        --------------
C:    00000009  _Producer                               testcoop
C:    00000033  _Consumer                               testcoop
C:    00000059  _main                                   testcoop
C:    00000065  __sdcc_gsinit_startup                   testcoop
C:    00000069  __mcs51_genRAMCLEAR                     testcoop
C:    0000006A  __mcs51_genXINIT                        testcoop
C:    0000006B  __mcs51_genXRAMCLEAR                    testcoop
C:    0000006C  _Bootstrap                              cooperative
C:    0000008A  _ThreadCreate                           cooperative
C:    00000117  _ThreadYield                            cooperative
C:    0000016F  _ThreadExit                             cooperative
```

Fig 2: Address of respective functions in cooperative and testcoop for reference.

1. *ThreadCreate(main)*

ThreadCreate for main form cooperative is called in during startup using Boostrap (refer Fig. 3)

```
testcoop.c   cooperative.c   preemptive.c   testpreempt.c   Makefile

void Bootstrap(void)
{
    /*
     * [TODO]
     * initialize data structures for threads (e.g., mask)
     *
     * optional: move the stack pointer to some known location
     * only during bootstrapping. by default, SP is 0x07.
     *
     * [TODO]
     *     create a thread for main; be sure current thread is
     *     set to this thread ID, and restore its context,
     *     so that it starts running main().
     */
    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) call made in Boostrap

```
operative.c   testcoop.map   testcoop.rst   cooperative.h   Makefile   cooperative.rst
                                        427 ;   ----------------------------------------
000065                                  428 __sdcc_gsinit_startup:
                                        429 ;   testcoop.c:114:   endasm;
000065 02 00 6C              [24]       430     LJMP    _Bootstrap
                                        431 ;   testcoop.c:115: }
000068 22                    [24]       432     ret
```

Fig. 4: _Boostrap call in testcoop at address 0065H

```
stcoop.map   testcoop.rst   cooperative.h   Makefile   cooperative.rst
                                        316 ;   cooperative.c:92: threadMask = 0x00;    // initialise
00006C 75 36 00              [24]       317     mov _threadMask,#0x00
                                        318 ;   cooperative.c:94: currentThread = ThreadCreate(main);
00006F 90 00 59              [24]       319     mov dptr,#_main
000072 12 00 8A              [24]       320     lcall   _ThreadCreate
000075 85 82 34              [24]       321     mov _currentThread,dpl
```

Fig. 5: ThreadCreate call in cooperative in _Boostrap at address 0072H

As can be seen from Fig. 5 ThreadCreate(main) is called at address 0072H, hence BreakPoint on 0072H is added.

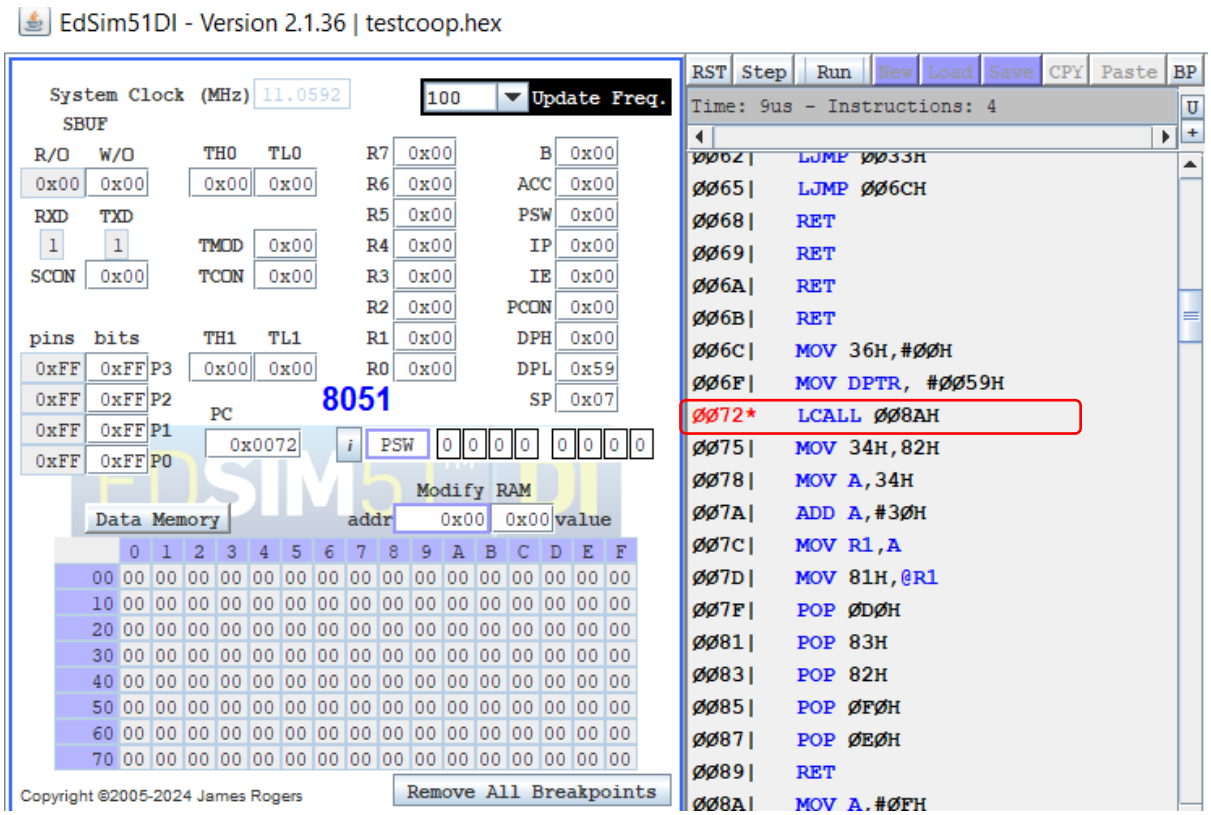


Fig. 6: Screenshot EdSim51 BreakPoint at 0072H

As shown in Figure 6, DPTR (i.e. DPH and DPL) is loaded with 0059 which is address of main in testcoop (refer Fig. 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 testcoop (refer Fig. 7), where Producer maps to 0009H (refer Fig. 2) and ThreadCreate call for Producer is at 005FH can be observed in Fig. 8.

```c
void main(void)
{
    /*
     * [TODO]
     * initialize globals
     */
    bufferFull = 0;
    /*
     * [TODO]
     * set up Producer and Consumer.
     * Because both are infinite loops, there is no loop
     * in this function and no return.
     */
    ThreadCreate(Producer);
    //ThreadCreate(Consumer);
    Consumer();
    //ThreadYield();
}
```

Fig. 7: ThreadCreate(Producer) call made in main

```
                                  413 ;   testcoop.c:97: bufferFull = 0;
000059 75 3D 00            [24]   414     mov _bufferFull,#0x00
                                  415 ;   testcoop.c:104: ThreadCreate(Producer);
00005C 90 00 09            [24]   416     mov dptr,#_Producer
00005F 12 00 8A            [24]   417     lcall    ThreadCreate
                                  418 ;   testcoop.c:106: Consumer();
```

Fig. 8: ThreadCreate(Producer) call indicating at address 005FH.

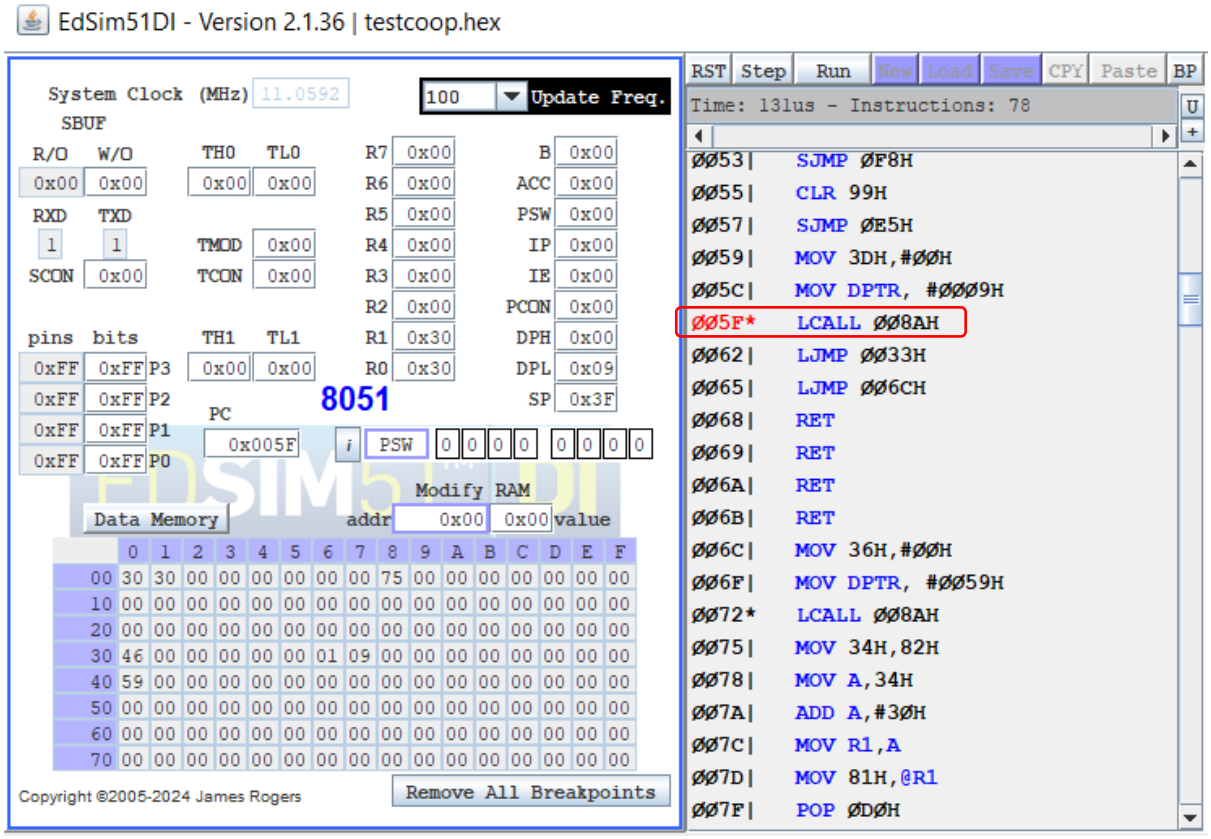


Fig. 9: Screenshot of EdSim51 for BreakPoint at 005FH (ThreadCreate(Producer))

As shown in Figure 9, DPTR (i.e. DPH and DPL) having 0009H which is Producer's address (refer Fig. 2), and SP is at 3FH. On the BreakPoint at 005FH it's going to call for 008AH which is address of ThreadCreate (refer Fig. 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. Variable "bufferFull" is a common variable for Producer and consumer to update, if sharedBuffer contains new character "bufferFull" is triggered to 1 and once character transferred SBUF "bufferFull" is set to 0.

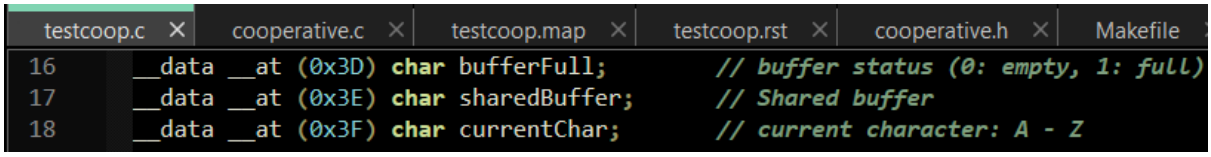


Fig. 10: Screenshot testcoop.c indicating address for respective variables

testcoop.c | cooperative.c | preemptive.c | testpreempt.c | Makefile

```c
void Producer(void)
{
    /*
     * [TODO]
     * initialize producer data structure, and then enter
     * an infinite loop (does not return)
     */
    currentChar = 'A';
    while (1)
    {
        /* [TODO]
         * wait for the buffer to be available,
         * and then write the new data into the buffer */
        while (bufferFull!=0) ThreadYield();
        sharedBuffer = currentChar;
        bufferFull = 1;
        currentChar = (currentChar == 'Z') ? 'A' : currentChar + 1;
        ThreadYield();
    }
}
```

Fig. 11: Produce code snippet

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

Data Memory addr 0x00 0x30 value

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	30	31	00	00	03	00	02	00	31	32	00	00	03	02	02	00
10	31	30	00	00	00	00	08	01	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	46	56	00	00	01	01	03	41	08	01	00	00	00	01	42	43
40	53	00	01	00	01	00	01	00	00	00	00	00	00	00	00	00
50	31	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 = “C” (i.e. 43H)
currentChar = “D” (i.e. 44H)

Data Memory addr 0x00 0x30 value

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	30	31	00	00	03	00	02	00	31	32	00	00	03	00	04	00
10	31	30	00	00	00	00	08	01	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	46	56	00	00	00	01	03	41	08	01	00	00	00	01	43	44
40	53	00	01	00	01	00	01	00	00	00	00	00	00	00	00	00
50	13	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)

Data Memory addr 0x00 0x30 value

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	30	31	00	00	03	00	02	00	31	32	00	00	03	00	04	00
10	31	30	00	00	00	00	08	01	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	46	56	00	00	01	01	03	41	08	01	00	00	00	01	44	45
40	53	00	01	00	01	00	01	00	00	00	00	00	00	00	00	00
50	13	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 = “E” (i.e. 45H)
currentChar = “F” (i.e. 46H)

Data Memory addr 0x00 0x30 value

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	30	31	00	00	03	00	02	00	31	32	00	00	03	00	04	00
10	31	30	00	00	00	00	08	01	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	46	56	00	00	01	01	03	41	08	01	00	00	00	01	45	46
40	53	00	01	00	01	00	01	00	00	00	00	00	00	00	00	00
50	13	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 10 indicating address of “sharedBuffer” and “currentChar” we can see in Table 2 that 3EH gets updated with value from 3FH and which is hex values of character “A” to “Z”.

Consumer in Run

When Consumer is running the SBUF receives the character from “sharedBuffer” and can be displayed on UART Receiver as shown in Figure 13 (a) and 13 (b) with code snippet in Fig. 12.

```
testcoop.c   cooperative.c   preemptive.c   testpreempt.c   Makefile

52  void Consumer(void)
53  {
54      /*
55       * [TODO]
56       * initialize Tx for polling
57       */
58      // Configure serial port for polling mode
59      TMOD = 0x20;    // Timer1 mode 2: 8-bit auto-reload
60      TH1 = 0xFA;     // (Hex) Baud rate 4800 for 11.0592 MHz or TH1=-6 //(
61      SCON = 0x50;    // Mode 1: 8-bit UART, REN enabled
62      TR1 = 1;        // Start Timer1
63
64      while (1)
65      {
66          /*
67           * [TODO]
68           * wait for new data from producer
69           */
70          while (bufferFull==0) ThreadYield();
71          /*
72           * [TODO]
73           * write data to serial port Tx,
74           * poll for Tx to finish writing (TI),
75           * then clear the flag
76           */
77          SBUF = sharedBuffer;
78          bufferFull = 0;
79          while (!TI) ThreadYield();  // Wait for transmission to complet(
80          TI = 0;        // Clear transmit interrupt flag
81
82      }
83  }
```

Fig. 12: Consumer code snippet

In Figure 13 (a), Consumer running with SBUF having W/O 0x42H (i.e. “B”) which will be the next character to displayed in UART receiver along with previous character (i.e. “A”).

And in Figure 13 (b), Consumer running with SBUF having W/O 0x49H (i.e. “I”) which will be the next character to displayed in UART receiver along with previous character (i.e. “ABCDEFGH”).

0x3DH (i.e. “bufferFull”) is switched to 0 during consumer in execution as per Consumer code snippet in Fig. 12, it can be seen in Fig. 13(a & b).

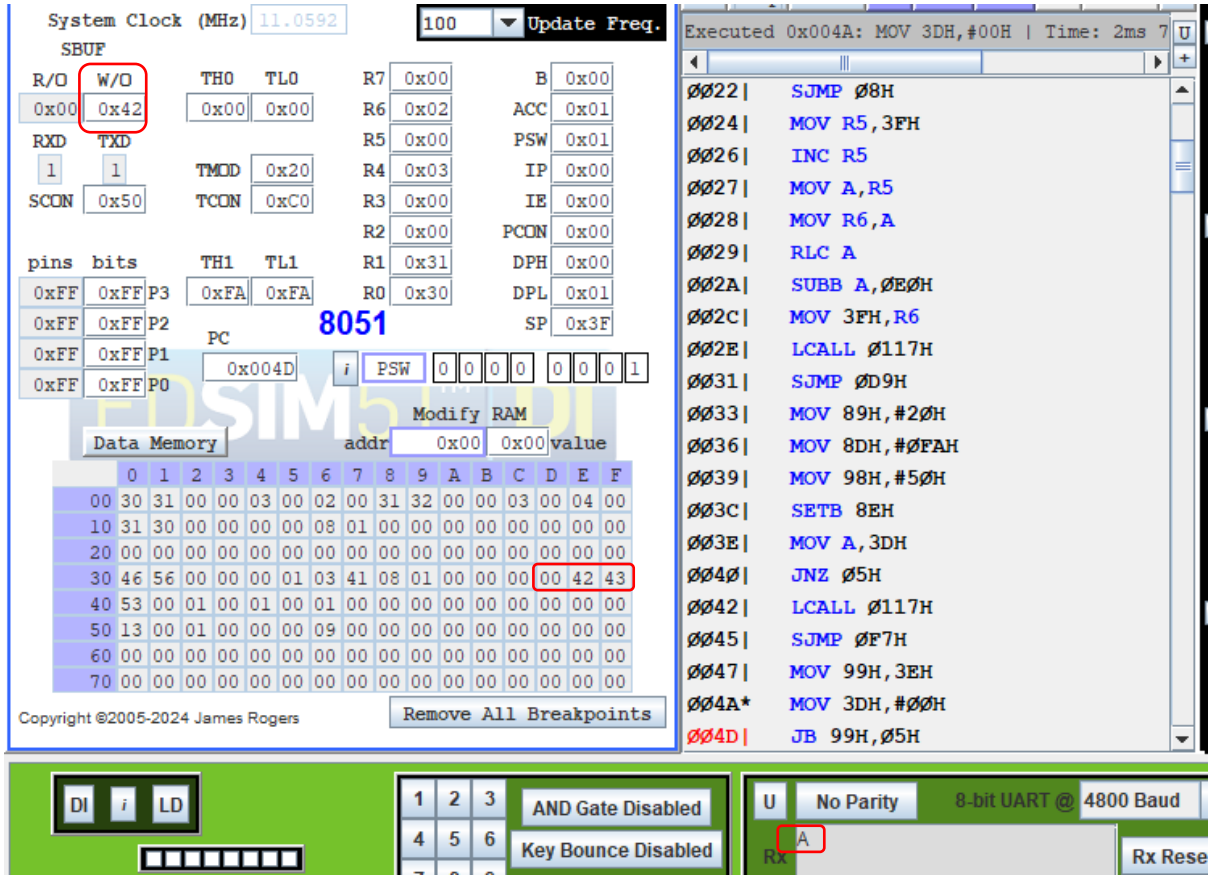


Fig. 13 (a): Screenshot (1) of EdSim51 while Consumer is running.

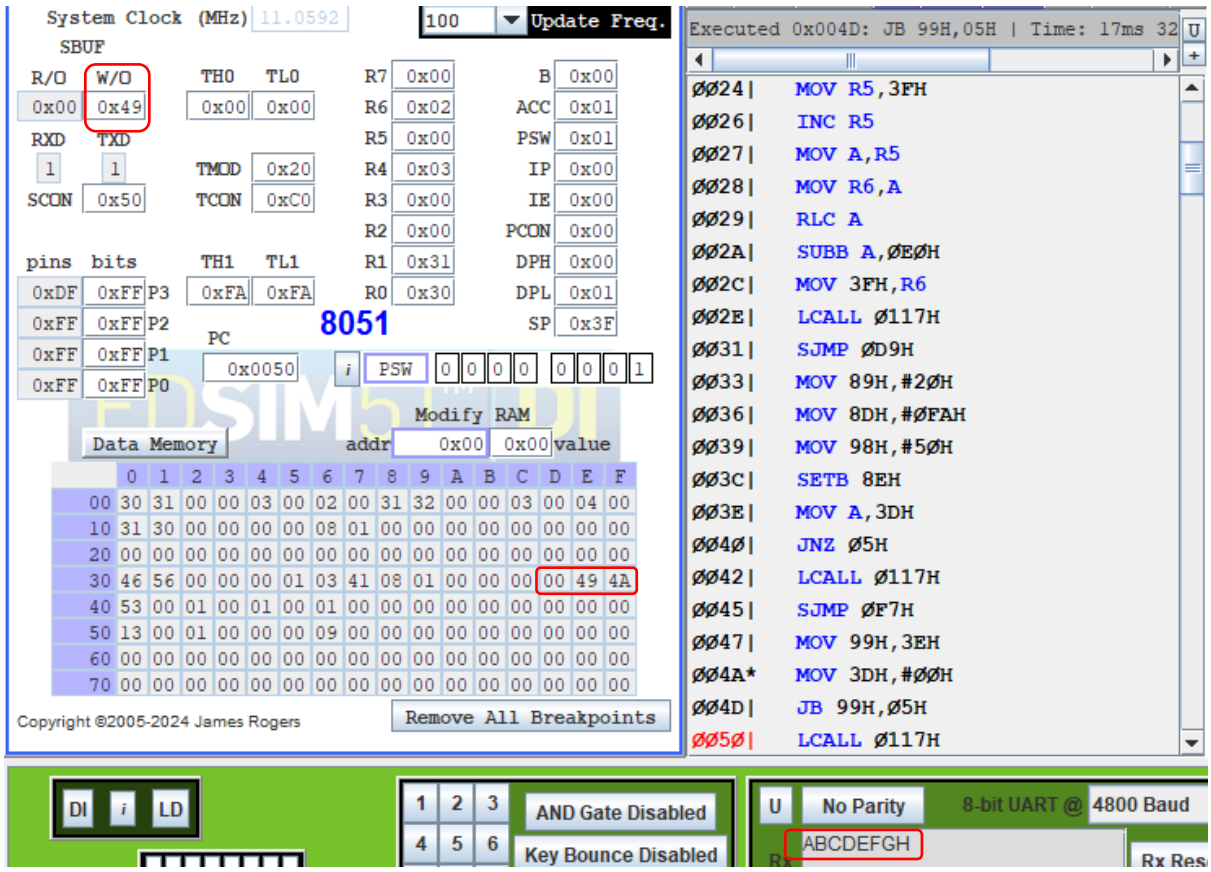


Fig. 13 (b): Screenshot (2) of EdSim51 while Consumer is running.