PPC 3

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 same Makefile from CP2 for CP3 since the file names mentioned in both are the same hence it's compatible. 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.

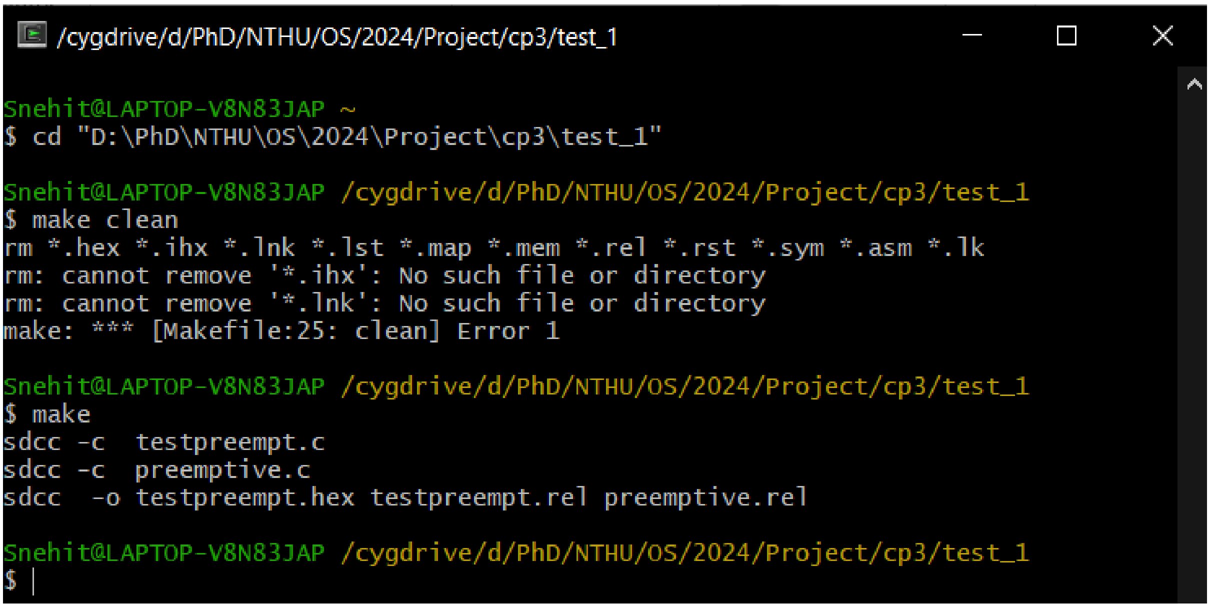


Fig. 1: Screenshot of Cygwin after running *make clean* and *make* command.

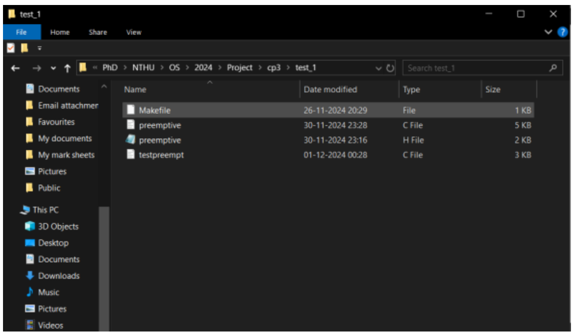
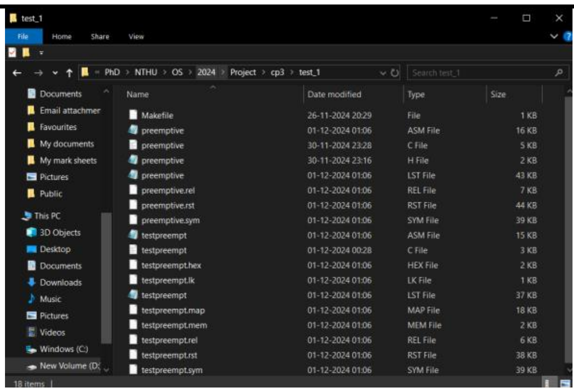
After *$ make clean*	**After *$ make***
	

Table 2: results of Makefile compilation

Mapping of variables and functions:

Value	Global	Global Defined
00000000	.__.ABS.	preemptive
00000027	_mutex	testpreempt
00000028	_full	testpreempt
00000029	_empty	testpreempt
0000002A	_head	testpreempt
0000002B	_tail	testpreempt
0000002C	_sharedBuffer	testpreempt
0000002F	_currentChar	testpreempt
00000030	_savedSP	preemptive
00000034	_currentThread	preemptive
00000035	_newThread	preemptive
00000036	_threadMask	preemptive

Fig. 3: Mapping of variables to respective memory locations.

Value	Global	Global Defined
00000014	_Producer	testpreempt
0000006D	_Consumer	testpreempt
000000BD	_main	testpreempt
000000E9	__sdcc_gsinit_startup	testpreempt
000000ED	__mcs51_genRAMCLEAR	testpreempt
000000EE	__mcs51_genXINIT	testpreempt
000000EF	__mcs51_genXRAMCLEAR	testpreempt
000000F0	_timer0_ISR	testpreempt
000000F4	_Bootstrap	preemptive
0000011A	_ThreadCreate	preemptive
000001B2	_ThreadYield	preemptive
00000215	_ThreadExit	preemptive
00000278	_myTimer0Handler	preemptive

Fig. 4: Mapping of functions to respective memory locations.

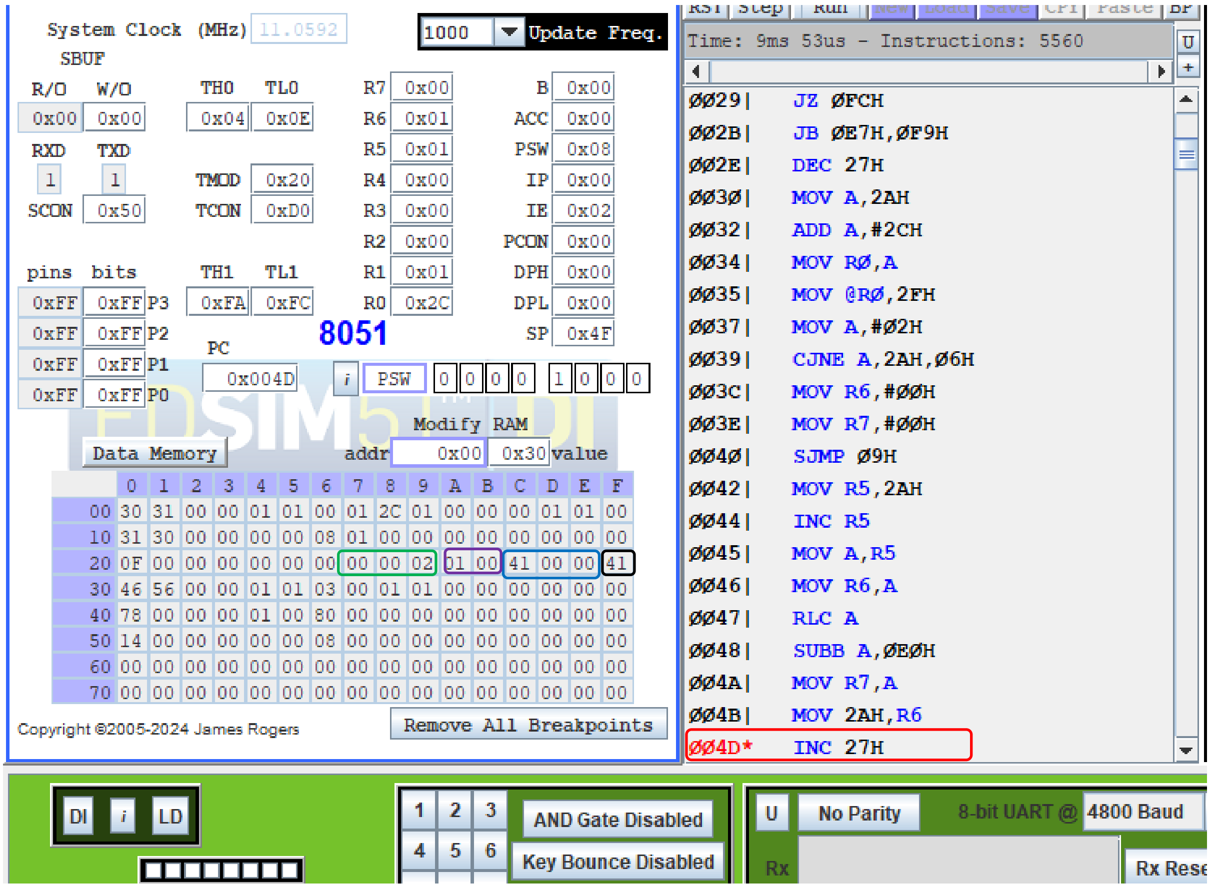
Producer in run:

The code snippet of Producer is as given in Fig. 5. Figure 6 shows the status of Semaphore and variable status during Producer in run. A breakpoint is added at address 004DH which corresponds SemaphoreSignal(mutex) in Producer.

```c
void Producer(void)
{
    currentChar = 'A';
    while (1)
    {
        SemaphoreWait(empty);
        __critical{
            SemaphoreWait(mutex);
            sharedBuffer[head] = currentChar;
            head = (head==BUFFER_SIZE-1) ? 0 : head+1;
            SemaphoreSignal(mutex);
        }
        SemaphoreSignal(full);
        currentChar = (currentChar == 'Z') ? 'A' : currentChar + 1;
    }
}
```

Fig. 5: Producer code snippet

It was said to declare 3-deep char buffer, Hence, we have variable sharedBuffer is of length 3. Variable head is used as index to assign value in sharedBuffer, e.g. from Table 3, head=01 (Fig. 6 (a)) is the index for next Character to be assigned at sharedBuffer i.e. sharedBuffer[1] (Fig. 6 (b)). And similarly for onward head values, once it reaches maximum index (i.e. 2) it'll reset to 0 and cycle repeats in the same way.



(a)

	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	2D	01	00	00	00	02	02	00
10	31	30	00	00	00	00	08	01	00	00	00	00	00	00	00	00
20	0F	00	00	00	00	00	00	00	01	01	02	00	41	42	00	42
30	46	56	00	00	01	01	03	00	01	01	00	00	00	00	00	00
40	78	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

(b)

	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	2E	01	00	00	00	43	00	00
10	31	30	00	00	00	00	08	01	00	00	00	00	00	00	00	00
20	0F	00	00	00	00	00	00	00	02	00	00	00	41	42	43	43
30	46	56	00	00	01	01	03	00	01	01	00	00	00	00	00	00
40	78	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

(c)

Fig 6: Producer in run with respective semaphore and variable changes till it full (from a to c).

Variable	Memory location	Value in Fig. 5 (a)	Value in Fig. 5 (b)	Value in Fig. 5 (c)
Mutex	0x27	00	00	00
Full	0x28	00	01	02
empty	0x29	02	01	00
head	0x2A	01	02	00
tail	0x2B	00	00	00
sharedBuffer[0]	0x2C	41	41	41
sharedBuffer[1]	0x2D	00	42	42
sharedBuffer[2]	0x2E	00	00	43
currentChar	0x2F	41	42	43

Table 3: Semaphore and variables change during producer run from Fig. 5 (a-c).

currentChar is variable for loop through character A-Z, the value of which is going to be assigned to sharedBuffer as per head index. Produce remain in run till Semaphore full reached maximum index possible (or say, till all the sharedBuffer get assigned to new values). Hence notice from Fig. 6 (c) the semaphore full is at 02 (at maximum) and sharedBuffer gets assigned to values 41, 42, 43 (i.e. character “A”, “B” and “C”).

For convenience Table 3 is given indicating necessary values of variables at mentioned memory locations during producer run which is taken from Fig. 6 (a -c).

Consumer in run:

After Producer completes its run with sharedBuffer filled with new set of characters, the Consumer is designed to transfer those characters to SBUF to UART. Code snippet of Consumer is given in Fig. 7.

Breakpoint at 00B9H is added which corresponds to the SemaphoreSignal(empty) during consumer is running. Which means before reaching this Breakpoint, the SBUF is updated with the character from sharedBuffer at index decided by tail from previous value. Consider Fig. 8 (a) previous value of tail is 0, sharedBuffer[tail] is 41H (i.e. “A”) which is assigned to SBUF at W/O, and it’ll be the character displayed in UART receiver (see Fig. 8 (b)).

Consumer will be in run till Semaphore empty reach maximum value (i.e. 2) since design buffer size is of length 3. Hence as empty gets updated in each cycle, the SBUF will gets assigned to characters from sharedBuffer with index from tail. Observe from 8 (a), (b) and (c), SBUF is assigned to 41, 42, and 43 in each cycle and this character will then transfer to UART receiver i.e. “A”, “B” and “C” respectively. Fig. 9 is after consumer is executed, shows all the characters in transferred to SBUF.

```c
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)
    {
        SemaphoreWait(full);
        __critical{
            SemaphoreWait(mutex);
            SBUF = sharedBuffer[tail];
            while (!TI);  // Wait for transmission to complete
            TI = 0;       // Clear transmit interrupt flag
            tail = (tail==BUFFER_SIZE-1) ? 0 : tail+1;
            SemaphoreSignal(mutex);
        }
        SemaphoreSignal(empty);
    }
}
```

Fig. 7: Consumer code snippet

For convenience Table 4 is given indicating necessary values of variables at mentioned memory locations during consumer run which is taken from Fig. 8 (a -c).

Variable	Memory location	Value in Fig. 6 (a)	Value in Fig. 6 (b)	Value in Fig. 6 (c)
Mutex	0x27	01	01	01
Full	0x28	02	01	00
empty	0x29	00	01	02
head	0x2A	00	00	00
tail	0x2B	01	02	00
sharedBuffer	0x2C - 0x2F	41, 42, 43	41, 42, 43	41, 42, 43
SBUF	0x99	41	42	43

Table 4: Semaphore, variables and SBUF change during consumer run from Fig. 6 (a-c).

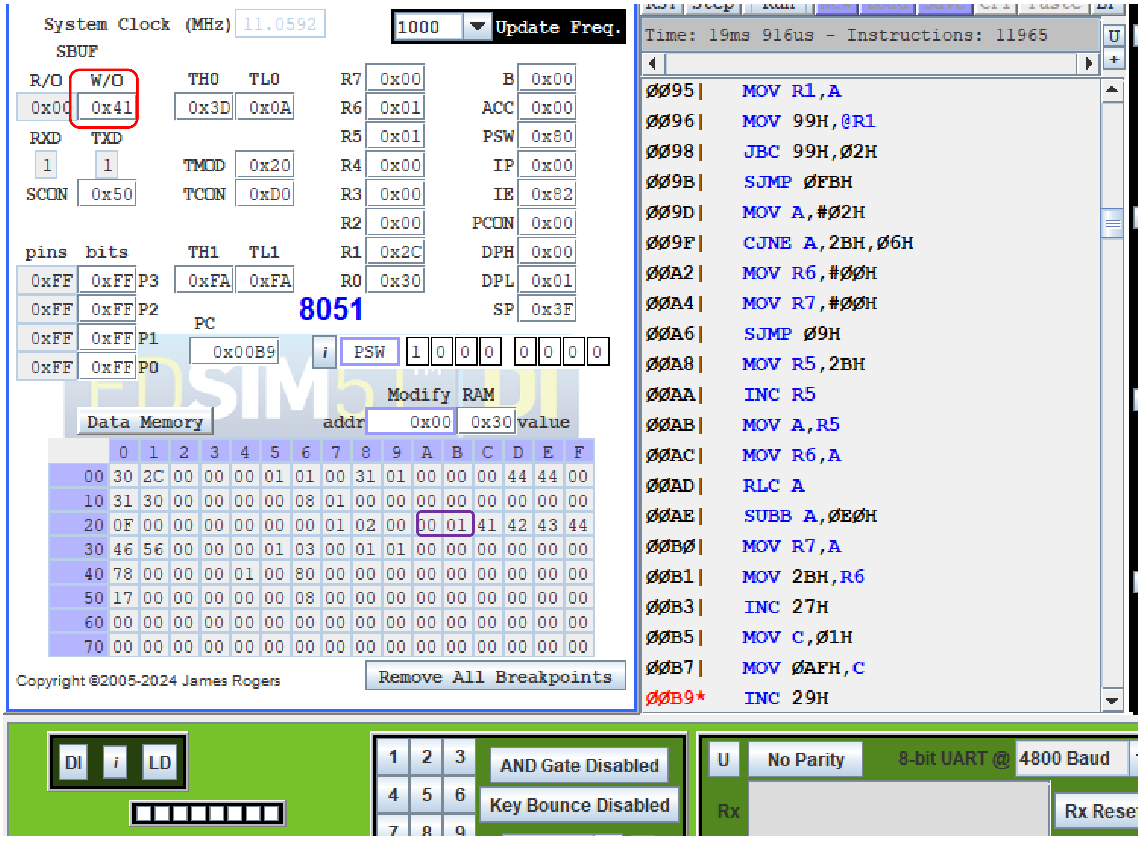


Fig 8 (a): Consumer in run with SBUF W/O at 0x41

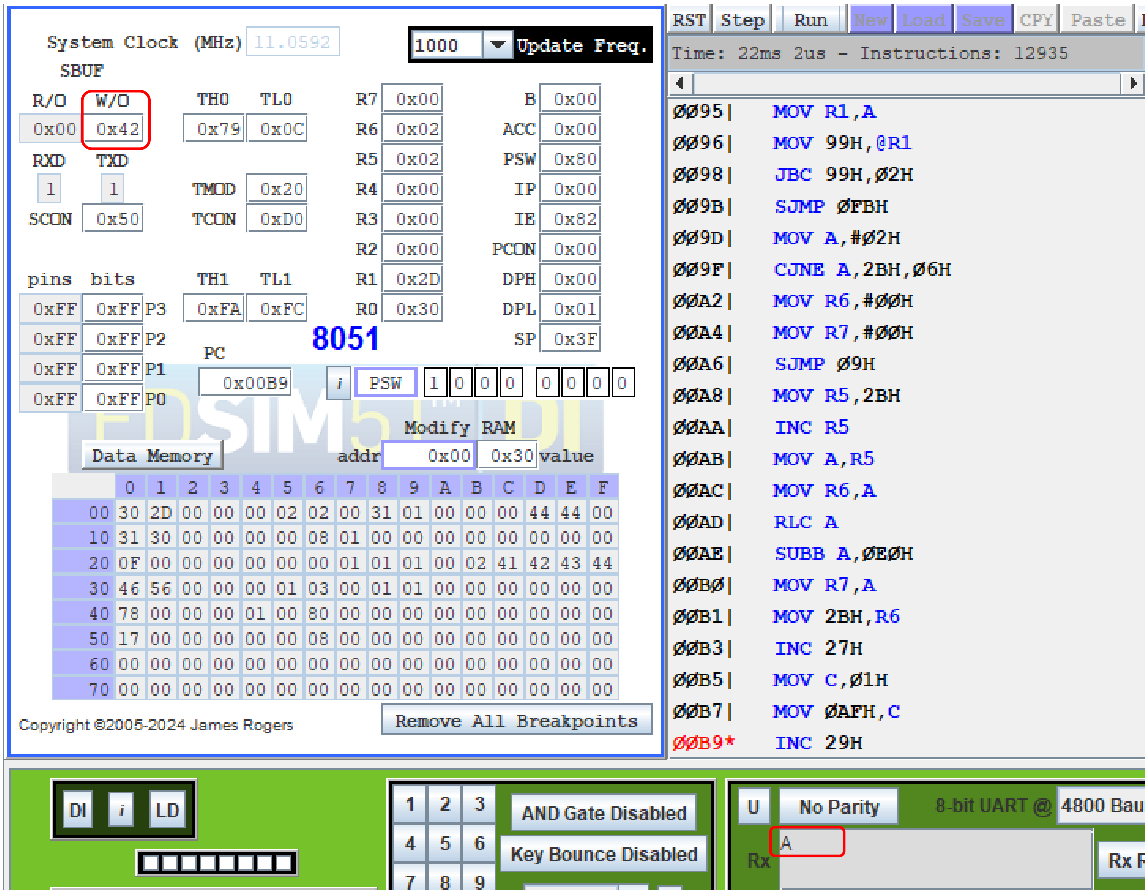


Fig 8 (b): Consumer in run with SBUF W/O at 0x42 and UART display A (i.e. 0x41)

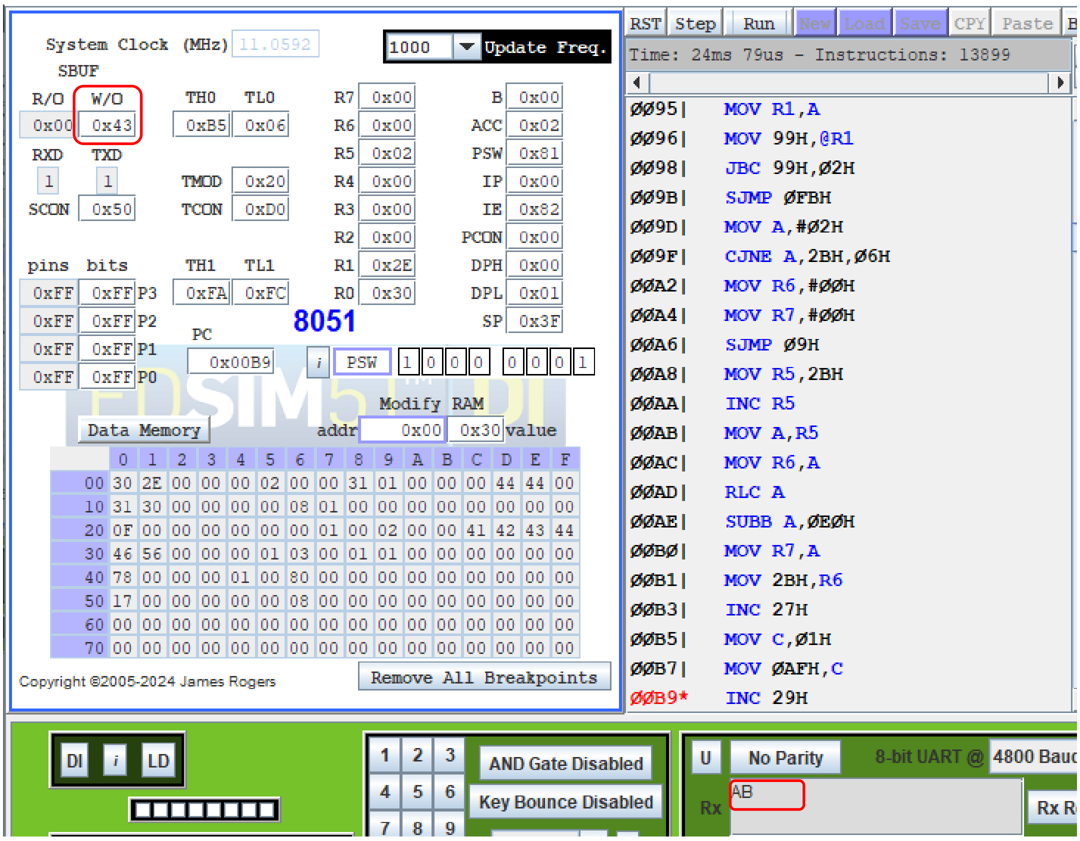


Fig 8 (c): Consumer in run with SBUF W/O at 0x43 and UART display AB (i.e. 0x41 and 0x42)

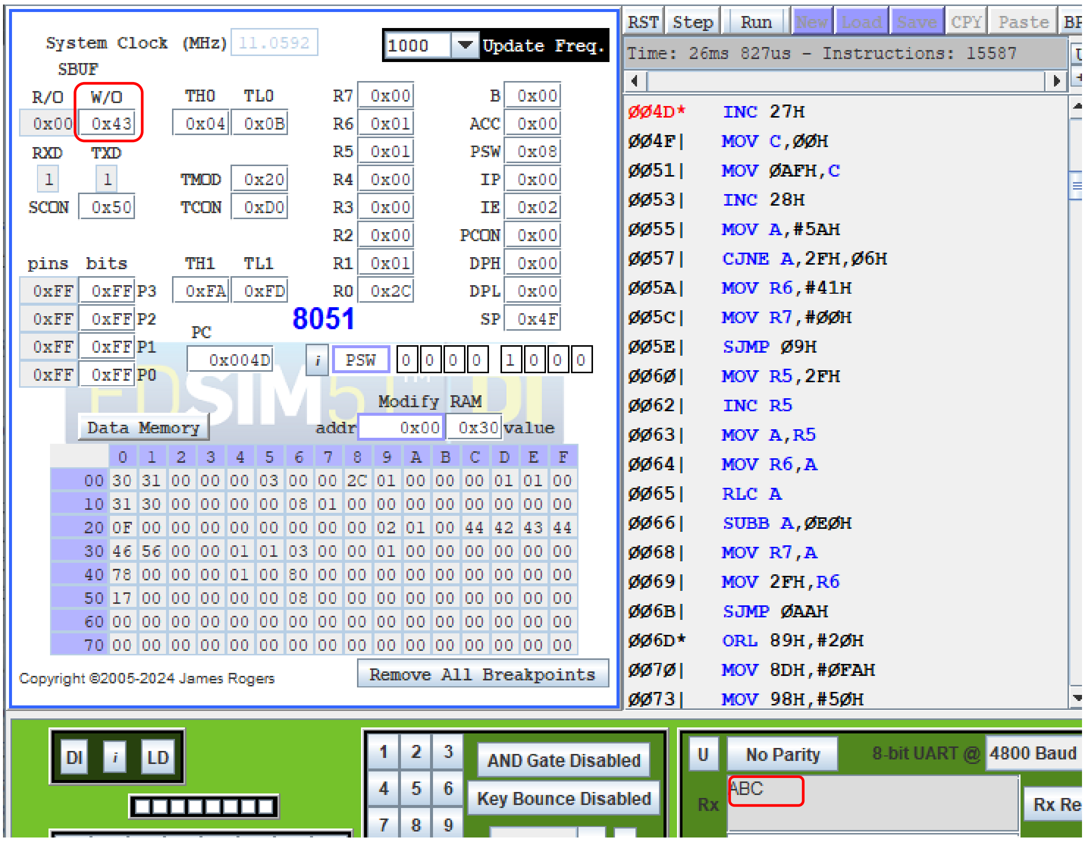


Fig 9: Consumer after execution with UART display ABC (i.e. 0x41, 0x42 and 0x43)

Onward the cycle is repeated with new characters assigned to sharedBuffer by Producer and Consumer forward it to SBUF and displayed on UART receiver which is shown in Figure 10.

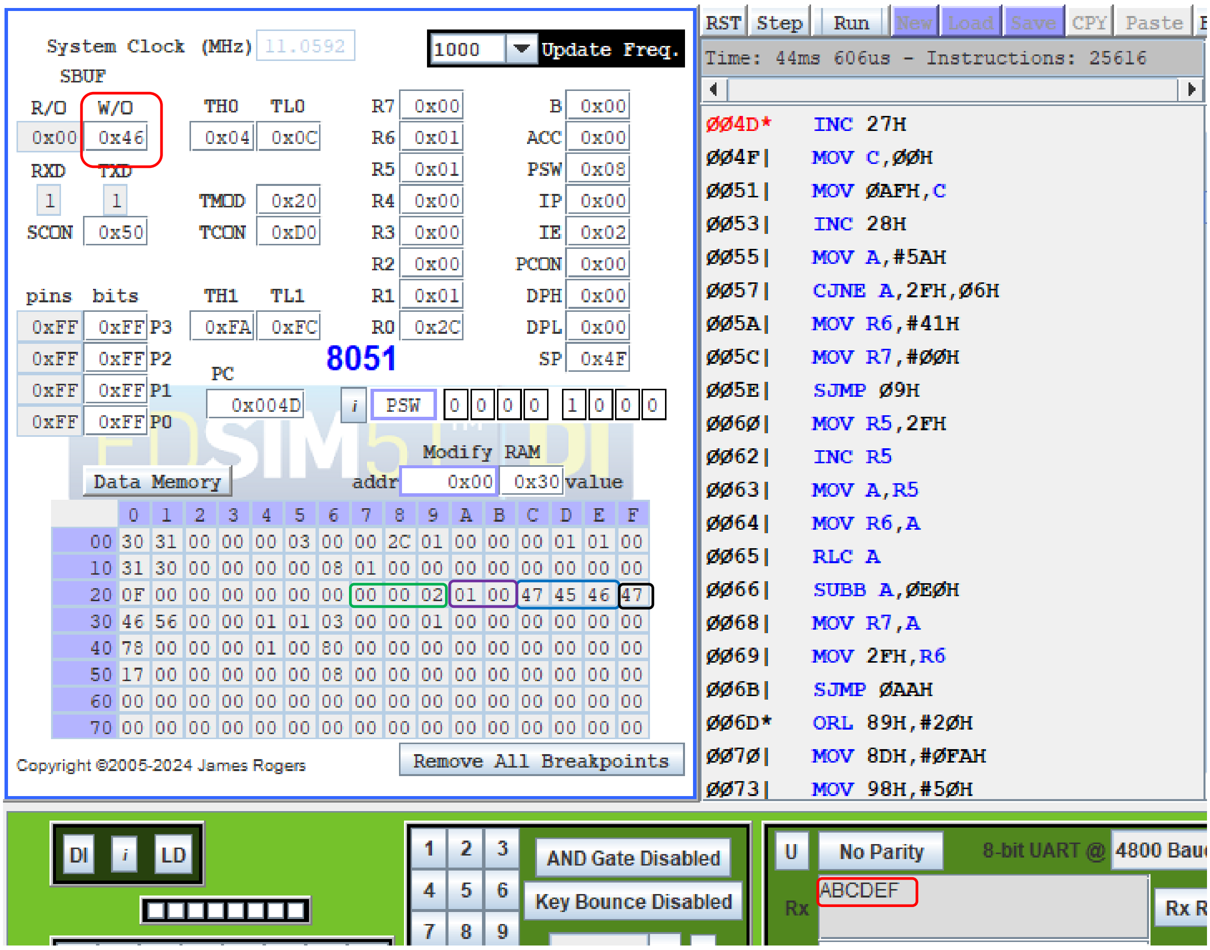


Fig 10: Sample showing the execution of producer for updated sharedBuffer and execution of consumer with latest characters on SBUF and UART