

# Code Explanation

## **testpreempt.c:**

I implemented this code similar to the testcoop.c so I would like to explain only the difference parts.

- For producer and consumer, I used the `__critical {}` to ensure my code is accessed atomically. By doing this I can remove the calling of Threadyield.

## **Preemptive.c:**

I implemented this code similar to the cooperative.c so I would like to explain only the difference parts.

- For initialize and enable timer 0, I follow the provided google docs
- Then I used 'EA' (Enable Interrupts) to control whether interrupts are globally enabled or disabled during critical sections of the program. So the format of the code using this EA will be something like this:

```
EA = 0; // Disable interrupts temporarily
// Critical section (e.g., thread creation, saving state)
EA = 1; // Enable interrupts after the critical section
```

By disabling interrupts, I can ensure that these operations aren't interrupted and after that it can resume and the system can handle tasks like context switching or responding to events like timers

- Then for my timer0handler, I follow the provided information from google docs including using RETI instead of RET (to return from the interrupt). My implementation is based on my ThreadYield function.

# Screenshots for compilation

```

(base) nattapat@Nattapats-MacBook-Pro 111006203_ppc2 % make clean
rm *.hex *.ihx *.lnk *.lst *.map *.mem *.rel *.rst *.sym *.asm *.lk
rm: *.ihx: No such file or directory
rm: *.lnk: No such file or directory
make: *** [clean] Error 1
(base) nattapat@Nattapats-MacBook-Pro 111006203_ppc2 % make
sdcc -c testpreempt.c
sdcc -c preemptive.c
preemptive.c:146: warning 85: in function ThreadCreate unreferenced function arg
ument : 'fp'
sdcc -o testpreempt.hex testpreempt.rel preemptive.rel
(base) nattapat@Nattapats-MacBook-Pro 111006203_ppc2 %

```

Figure 1 Screenshot for compilation

# Screenshots and explanation

```

testpreempt.map
00000085 _P0_5      preemptive
00000086 _P0_6      preemptive
00000087 _P0_7      preemptive
00000088 _PCON      preemptive
00000089 _IT0       preemptive
0000008A _TCON      preemptive
0000008B _IE0       preemptive
0000008C _TMO0      preemptive
0000008D _IT1       preemptive
0000008E _TL0       preemptive
0000008F _IE1       preemptive
00000090 _TL1       preemptive
00000091 _TH0       preemptive
00000092 _TR0       preemptive
00000093 _TF0       preemptive
00000094 _TH1       preemptive
00000095 _TR1       preemptive
00000096 _TF1       preemptive
00000097 _P1        preemptive
00000098 _P1_0      preemptive
00000099 _P1_1      preemptive
0000009A _P1_2      preemptive
0000009B _P1_3      preemptive
0000009C _P1_4      preemptive
0000009D _P1_5      preemptive
0000009E _P1_6      preemptive
0000009F _P1_7      preemptive
000000A0 _RT        preemptive
000000A1 _SCON      preemptive

```

Figure 2 testcoop.map (will be used for explanation in the following section)

```

ASxxxx Linker V03.00/V05.40 + $dld, page 13.
Hexadecimal [32-Bits]

Area          Addr          Size          Decimal Bytes (Attributes)
-----
CSEG          00000014      00000318 =    792. bytes (REL,CN,CODE)

Value Global                                Global Defined In Module
-----
C: 00000014 _Producer                      testpreempt
C: 00000049 _Consumer                      testpreempt
C: 0000007A _main                          testpreempt
C: 00000089 _sdcc_qsinit_startup            testpreempt
C: 0000008D _mc451_genRAMCLEAR             testpreempt
C: 0000008E _mc451_genXINIT                testpreempt
C: 0000008F _mc451_genXBANCLEAR            testpreempt
C: 00000090 _timer0_ISR                    testpreempt
C: 00000094 _Bootstrap                     preemptive
C: 000000BC _ThreadCreate                  preemptive
C: 0000016F _ThreadYield                    preemptive
C: 000001D6 _ThreadExit                     preemptive
C: 00000248 _myTimer0Handler                preemptive
C: 000002A9 _moduint                       _moduint
C: 000002F6 _moduint                       _moduint

ASxxxx Linker V03.00/V05.40 + $dld, page 14.
Files Linked [ module(s) ]
testpreempt.rel [ 1 ]

```

Figure 3 testcoop.map (will be used for explanation in the following section)

Before each ThreadCreate call

- ThreadCreate(main), Address of main is 0x7A, as shown in figure 3

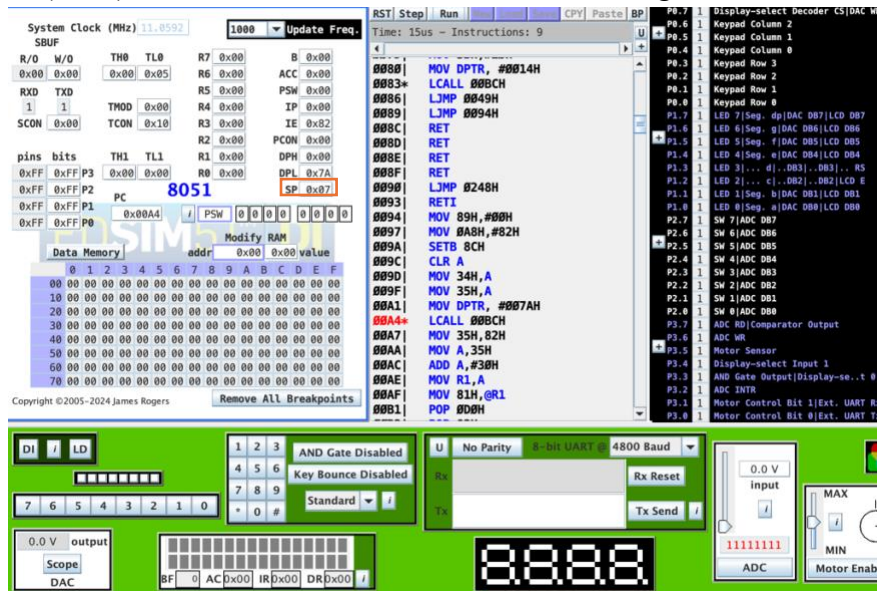


Figure 4 ThreadCreate(main)

When perform LCALL, the DPTR with the value 0x7A (which is the address of the main) the return address is pushed into the stack 2 bytes which leads into the changes of SP from 0x07 -> 0x09.

- ThreadCreate(Producer), Address of producer is 0x14, as shown in figure 3

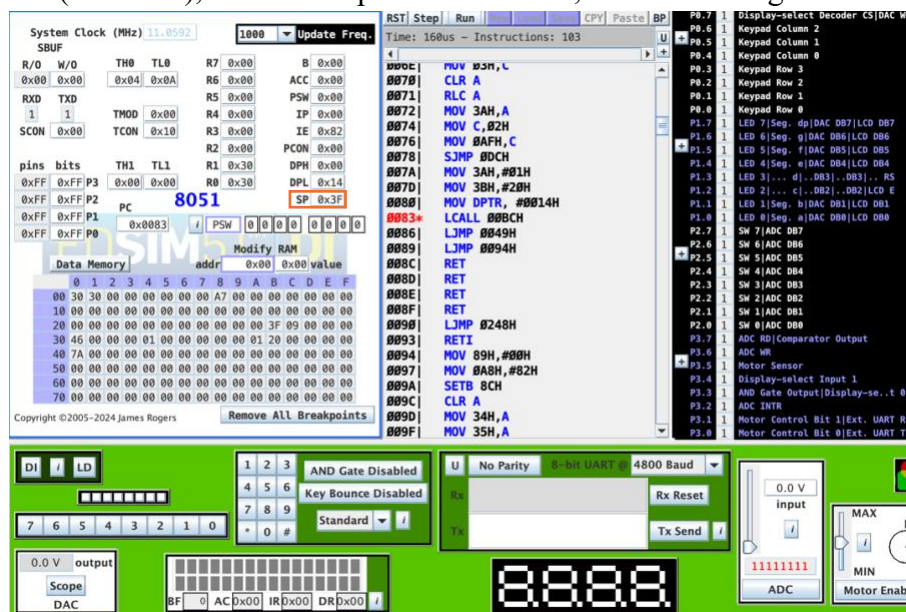


Figure 5 ThreadCreate(Producer)

When perform LCALL, the DPTR with the value 0x14 (which is the address of the producer), the return address is pushed into the stack 2 bytes which leads into the changes of SP from 0x3F -> 0x41.

When the producer is running

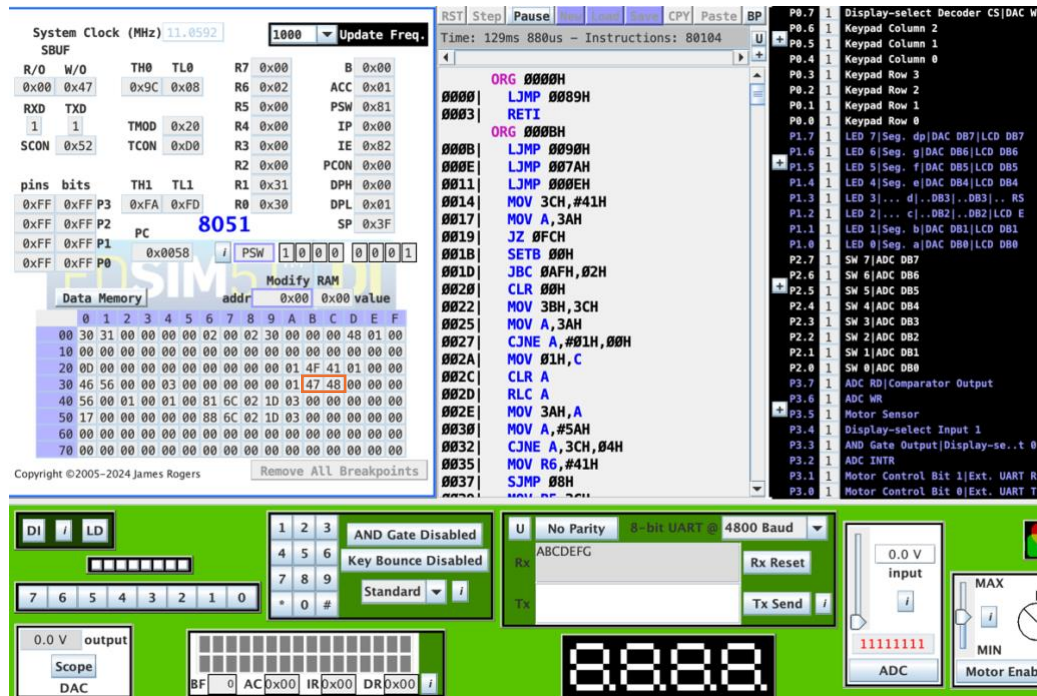


Figure 6 Producer (running)

As I have set *nextChar* (indicate which character will be transmit into SBUF) and *buffer* (represent shared buffer) in the address of 0x3B and 0x3C, respectively. Since producer is the one who generates a character and keep it in the shared buffer, therefore if the values in those addresses change that means the producer is running. (And it's changing over time)

When the consumer is running

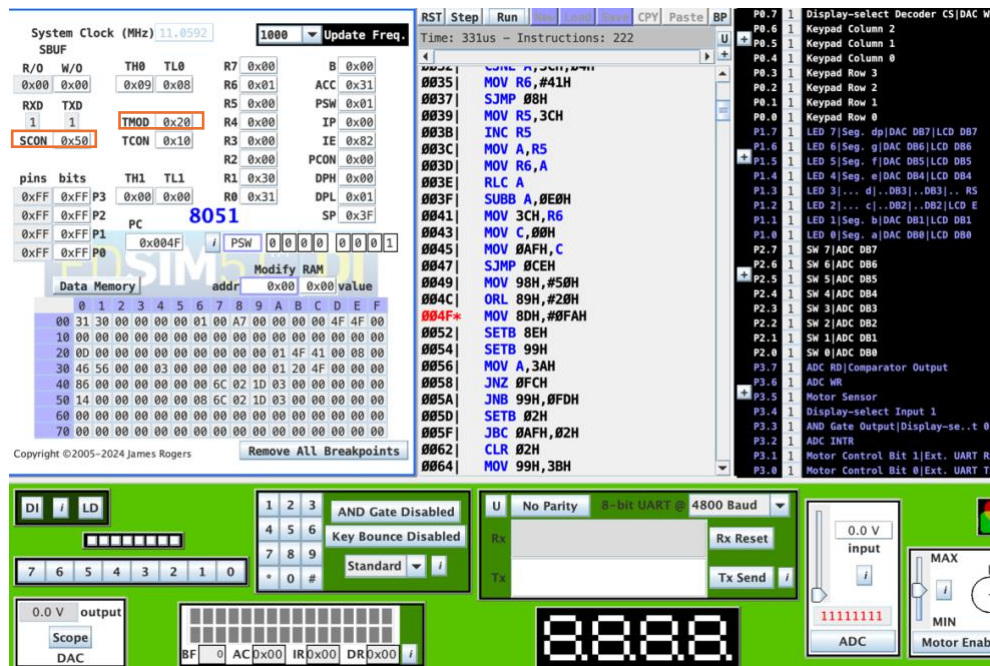


Figure 7 Consumer (running)

When the consumer is running, the value of TMOD and SCON will be changed to what we set in the code, which is (TMOD = 0x20, preserves the Bootstrap code's setting) and 0x50, respectively.

How can you tell that the interrupt is triggering on a regular basis?

Yes, the interrupt triggers regularly because When TH0 and TL0, Timer 0 (TMOD = 0, in bootstrap) reaches its **maximum count and overflows** (goes back to 0), it automatically triggers the interrupt, calling the function specified for handling Timer 0 interrupts. This invokes myTimer0Handler, which performs thread switching. The regular execution of threads confirms the periodic interrupts