

Programming Project Checkpoint 2

111062311 林哲宇

1. Screenshot for compilation:

```
jeffreylin0909@DESKTOP-Q29MBHF:/mnt/c/Users/林哲宇/OneDrive/桌面/OS/OS check point 2/ppc2$ make
sdcc -c testpreempt.c
testpreempt.c:65: warning 158: overflow in implicit constant conversion
sdcc -c preemptive.c
preemptive.c:206: warning 85: in function ThreadCreate unreferenced function argument : 'fp'
sdcc -o testpreempt.hex testpreempt.rel preemptive.rel
jeffreylin0909@DESKTOP-Q29MBHF:/mnt/c/Users/林哲宇/OneDrive/桌面/OS/OS check point 2/ppc2$ ls
Makefile      preemptive.h    preemptive.rst  testpreempt.c   testpreempt.lst  testpreempt.rel
preemptive.asm preemptive.lst  preemptive.sym  testpreempt.hex  testpreempt.map  testpreempt.rst
preemptive.c   preemptive.rel  testpreempt.asm testpreempt.lk   testpreempt.mem  testpreempt.sym
jeffreylin0909@DESKTOP-Q29MBHF:/mnt/c/Users/林哲宇/OneDrive/桌面/OS/OS check point 2/ppc2$
```

→ The warning message is because of setting TH1 to negative number (-6 for UART baud rate), and it's totally safe.

→ The warning message is because of using DPH/DPL instead of identifier of parameter, and it's totally safe.

→ Files generated after compilation (including .hex and .map).

Note: for better understanding for following explanation, here's the variable address map of my code:

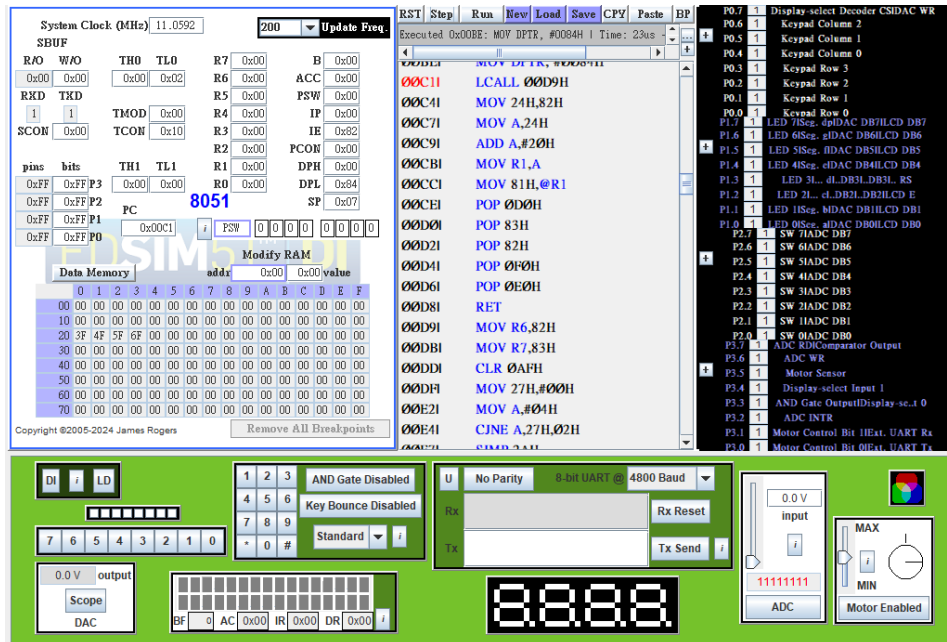
00000020	_T_SP (20~23,1 address space each entry)	cooperative
00000024	_current_T	cooperative
00000025	_tmp0	cooperative
00000026	_tmp1	cooperative
00000027	_tmp2	cooperative
00000028	_bitmap	cooperative
0000002C	_b_start	testcoop
0000002D	_b_end	testcoop
0000002E	_in_counter	testcoop
00000030	_buffer (30~3F,1 address space each entry)	testcoop

And here's the function address map:

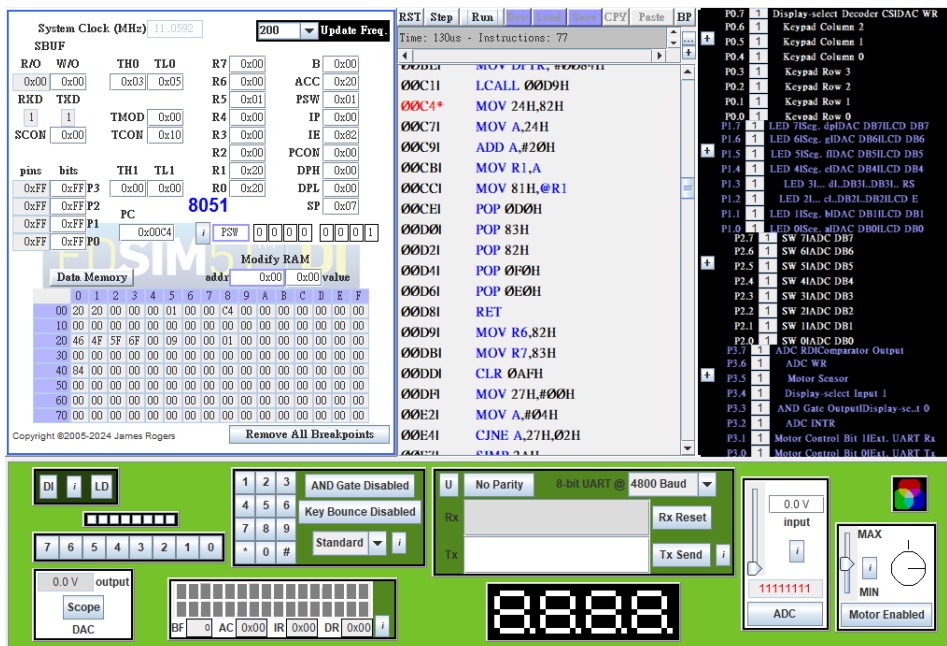
00000014	_Producer	testpreempt
00000053	_Consumer	testpreempt
00000084	_main	testpreempt
0000009C	__sdcc_gsinit_startup	testpreempt
000000A0	__mcs51_genRAMCLEAR	testpreempt
000000A1	__mcs51_genXINIT	testpreempt
000000A2	__mcs51_genXRAMCLEAR	testpreempt
000000A3	_timer0_ISR	testpreempt
000000A7	_Bootstrap	preemptive
000000D9	_ThreadCreate	preemptive
0000016B	_ThreadYield	preemptive
000001C0	_ThreadExit	preemptive
00000219	_myTimer0Handler	preemptive

2. Screenshots and explanation:

Before **ThreadCreate (main)** :



After **ThreadCreate (main)** :

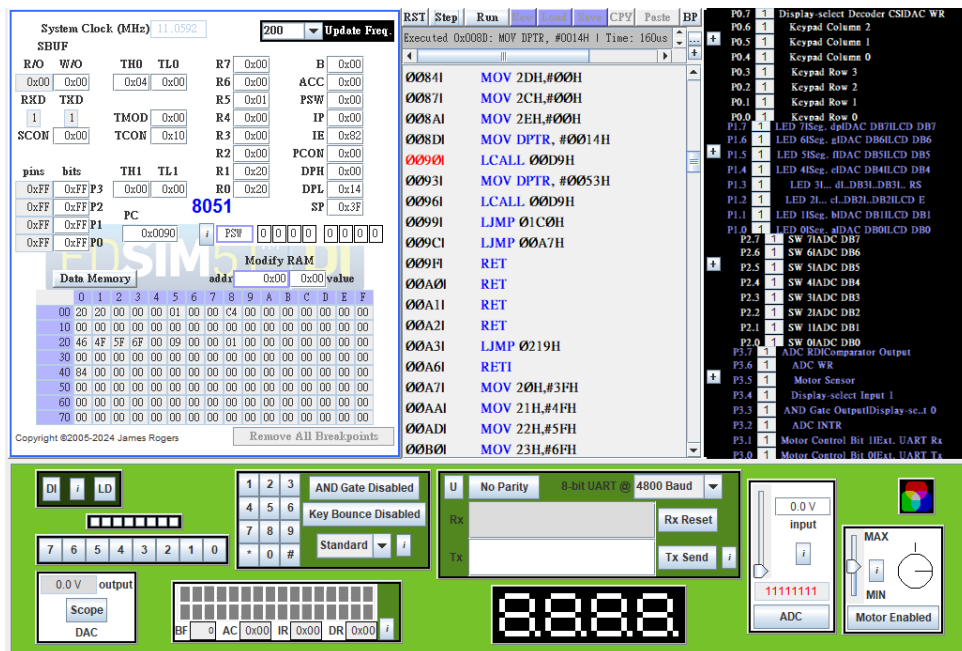


We can see that after **ThreadCreate (main)** , we can see:

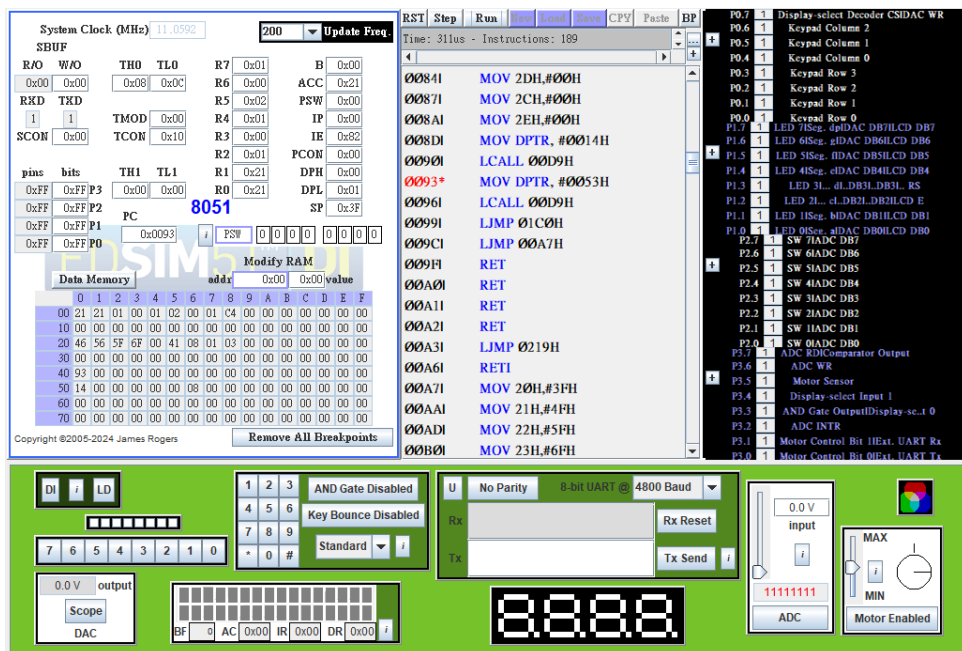
- **SP of thread #0 (address: 0x20) is changed to 46**
- **the bottom of stack of thread #0 (address: 0x40~0x41) is changed to 0084, which is the address of main function**
- **thread bitmap (address: 0x28) is also changed from 0000 to 0001**

meaning main function is assigned to thread #0.

Before ThreadCreate (Producer):



After ThreadCreate (Producer):



We can see that after **ThreadCreate (Producer)**, we can see:

- **SP of thread #1 (address: 0x21) is changed to 56**
- **the bottom of stack of thread #1 (address: 0x50~0x51) is changed to 0014, which is the address of Producer function**
- **thread bitmap (address: 0x28) is also changed from 0001 to 0011**

meaning Producer function is assigned to thread #1.

Before ThreadCreate (Consumer) :

The screenshot shows the Proteus IDE with the 8051 microcontroller. The CPU window displays the program counter (PC) at 0x0096 and the stack pointer (SP) at 0x3F. The data memory window shows the stack starting at 0x0096. The hardware window shows the DAC output at 0.0V and the ADC input at 11111111.

After ThreadCreate (Consumer) :

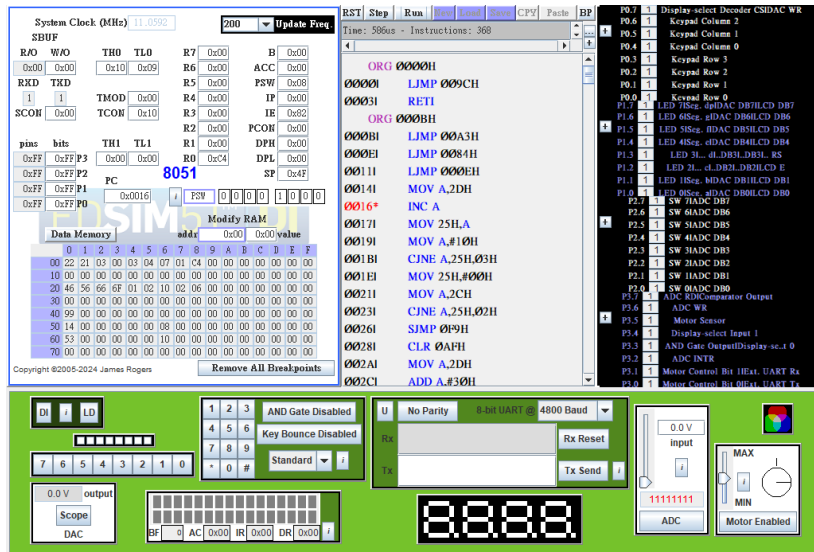
The screenshot shows the Proteus IDE with the 8051 microcontroller after ThreadCreate (Consumer). The CPU window displays the program counter (PC) at 0x0099 and the stack pointer (SP) at 0x22. The data memory window shows the stack starting at 0x0099. The hardware window shows the DAC output at 0.0V and the ADC input at 11111111.

We can see that after **ThreadCreate (Consumer)** , we can see:

- **SP of thread #2 (address: 0x22) is changed to 66**
- **the bottom of stack of thread #2 (address: 0x60~61) is changed to 0053, which is the address of Consumer function**
- **thread bitmap (address: 0x28) is also changed from 0011 to 0111**

meaning Consumer function is assigned to thread #2.

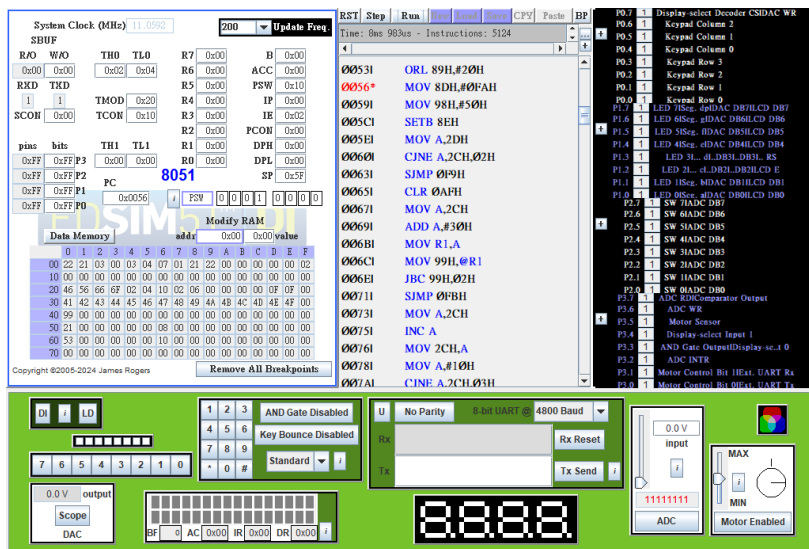
One screenshot when the Producer is running:



How do I know:

- Precise: since PC is in the code section of the function ($0x14 \leq PC < 0x53$).
- Roughly (ignore thread management function): Current thread number (address: 0x24) is 1.

One screenshot when the Consumer is running:



How do I know:

- Precise: since PC is in the code section of the function ($0x53 \leq PC < 0x84$).
- Roughly (ignore thread management function): Current thread number (address: 0x24) is 2.

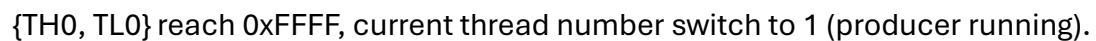
Every time {TH0, TL0} reach 0xFFFF, current thread number (address: 0x24) changes, means that interrupt (to switch current thread) is triggered on a regular basis.

The screenshot shows the Proteus simulator interface. The CPU window displays the assembly code for the initial state. The I/O window shows the DAC output as 0.0V and the ADC input as 11111111. The timer window shows the current thread number as 1.

Initially, current thread number is 1 (producer running).

The screenshot shows the Proteus simulator interface after the first interrupt. The CPU window displays the assembly code for the initial state. The I/O window shows the DAC output as 0.0V and the ADC input as 11111111. The timer window shows the current thread number as 2.

{TH0, TL0} reach 0xFFFF, current thread number switch to 2 (consumer running).



{TH0, TL0} reach 0xFFFF, current thread number switch to 1 (producer running).