鍾維聖 110062261

Programming Project Checkpoint 1 Report

Typescript for Compiling

```
wilbertallen@MacBook-Air-2016 ppc1 % make clean
rm *.hex *.ihx *.lnk *.lst *.map *.mem *.rel *.rst *.sym
rm: *.ihx: No such file or directory
rm: *.lnk: No such file or directory
make: *** [clean] Error 1
wilbertallen@MacBook-Air-2016 ppc1 % make
sdcc -c  testcoop.c
sdcc -c  cooperative.c
cooperative.c:213: warning 85: in function ThreadCreate unreferenced function argument : 'fp'
sdcc  -o testcoop.hex testcoop.rel cooperative.rel
wilbertallen@MacBook-Air-2016 ppc1 %
```

Fig.1 Typescript for compiling using the given makefile

Before Each ThreadCreate Call

Based on my understanding, ThreadCreate is called 2 times, one is for ThreadCreate(main), and the other is for ThreadCreate(Producer)

ThreadCreate(main);

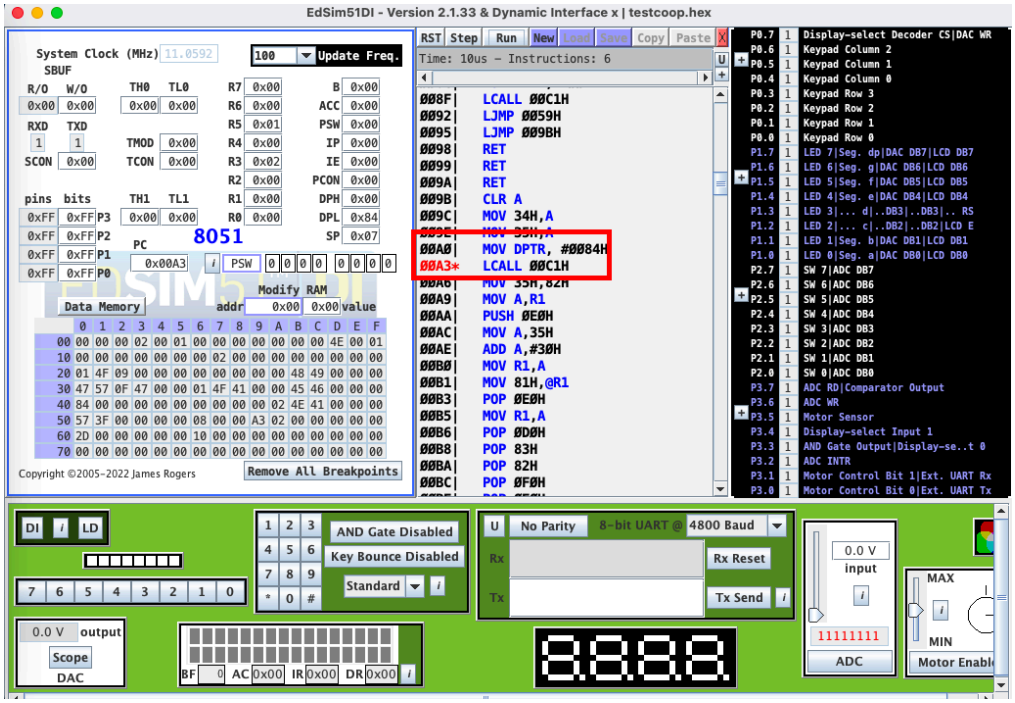


Fig2.1 Screenshot before ThreadCreate(main)

When the breakpoint reached LCALL, we can see that the address of main which is 0x84 can be seen in the DPL. It will then be pushed into SP, which will change 0x07 to 0x09.

ThreadCreate(Producer);

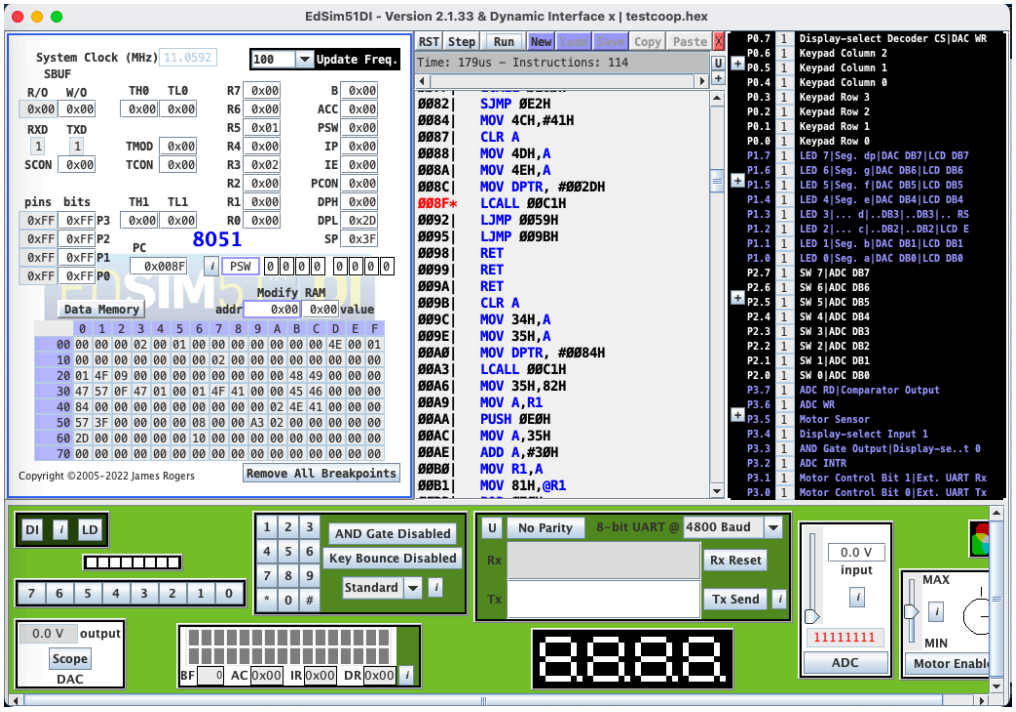
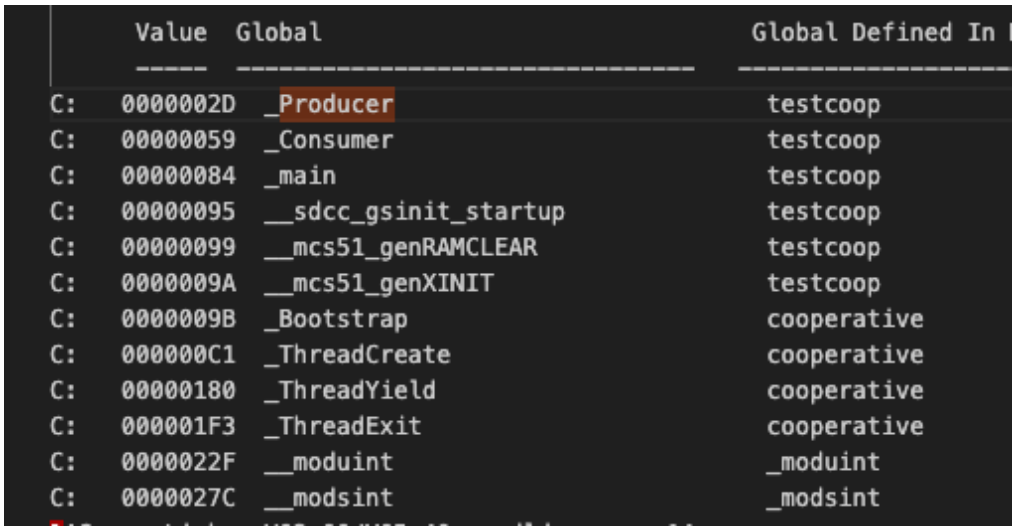


Fig.2.2 Screenshot before ThreadCreate(Producer)

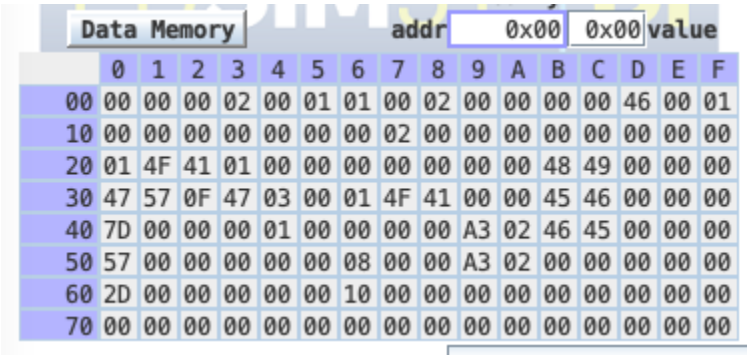
Same goes for ThreadCreate(Producer). At the breakpoint, the address of Producer which is 0x2D is pushed into the Stack Pointer, which will result in a change in the SP from 0x3F to 0x41.



	Value	Global	Global Defined In
C:	0000002D	_Producer	testcoop
C:	00000059	_Consumer	testcoop
C:	00000084	_main	testcoop
C:	00000095	__sdcc_gsinit_startup	testcoop
C:	00000099	__mcs51_genRAMCLEAR	testcoop
C:	0000009A	__mcs51_genXINIT	testcoop
C:	0000009B	_Bootstrap	cooperative
C:	000000C1	_ThreadCreate	cooperative
C:	00000180	_ThreadYield	cooperative
C:	000001F3	_ThreadExit	cooperative
C:	0000022F	__moduint	_moduint
C:	0000027C	__modsint	_modsint

Fig.2.3 Function addresses value

Running Producer

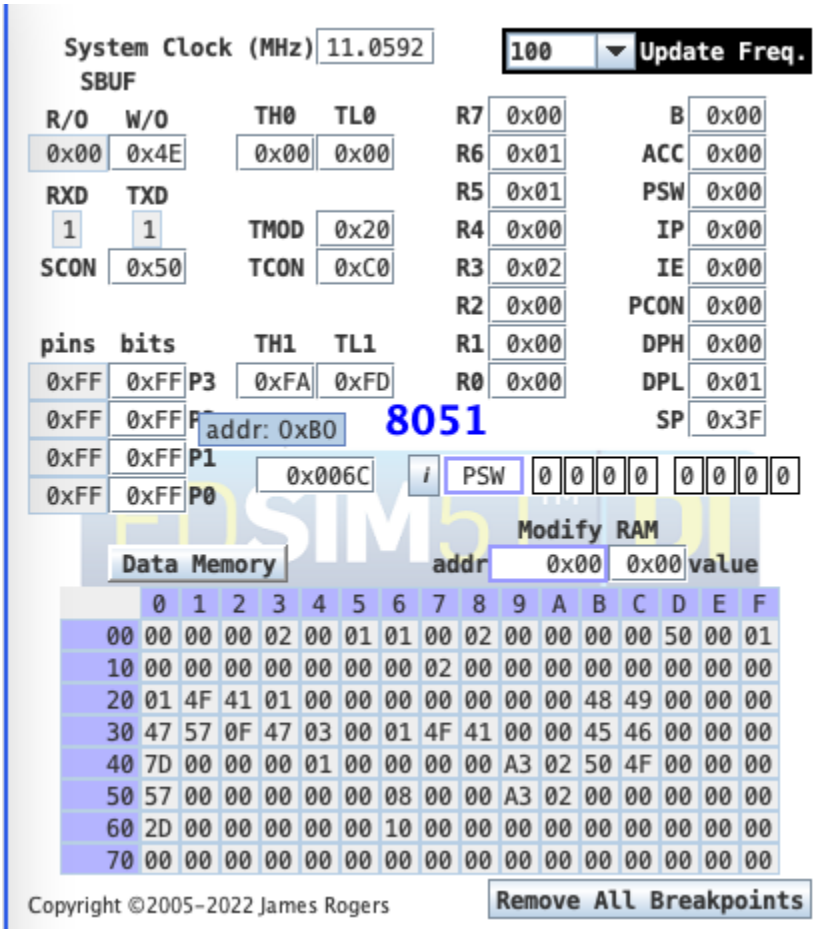


	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	00	00	00	02	00	01	01	00	02	00	00	00	00	46	00	01
10	00	00	00	00	00	00	00	02	00	00	00	00	00	00	00	00
20	01	4F	41	01	00	00	00	00	00	00	00	48	49	00	00	00
30	47	57	0F	47	03	00	01	4F	41	00	00	45	46	00	00	00
40	7D	00	00	00	01	00	00	00	00	A3	02	46	45	00	00	00
50	57	00	00	00	00	00	08	00	00	A3	02	00	00	00	00	00
60	2D	00	00	00	00	00	10	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

We can see that the Producer function is running by observing the addresses of *shared_buff* and *next_buff*. In my implementation, I store *shared_buff* on 0x4C and *next_buff* on 0x4B. At each iteration, the value from 0x4B will be copied to 0x4C. At the above example, we know that the current value of the shared buffer is 45 which is HEX for the ASCII character 'E', and the next buffer has a value of 46, which translates to 'F'.

Link of additional producer video : Drive

Running Consumer



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	00	00	00	02	00	01	01	00	02	00	00	00	00	50	00	01
10	00	00	00	00	00	00	00	02	00	00	00	00	00	00	00	00
20	01	4F	41	01	00	00	00	00	00	00	00	48	49	00	00	00
30	47	57	0F	47	03	00	01	4F	41	00	00	45	46	00	00	00
40	7D	00	00	00	01	00	00	00	00	A3	02	50	4F	00	00	00
50	57	00	00	00	00	00	08	00	00	A3	02	00	00	00	00	00
60	2D	00	00	00	00	00	10	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

We can know Consumer is running by observing the SBUF. SBUF is currently writing out 0x4E to the received_data part on Edsim, while the shared_buffer is storing the value for the next character to write out, which is 0x4F.