Programming Project Checkpoint 1

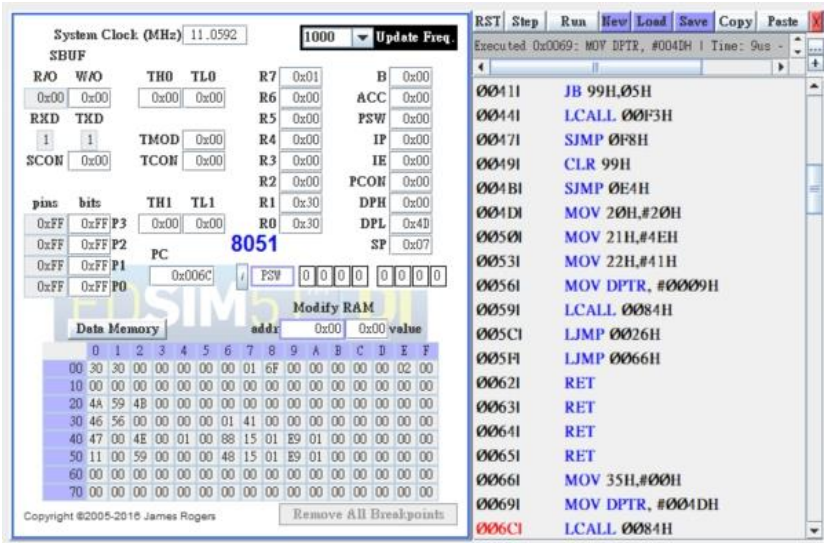
IPNS 106010018 Yen-Fong Li

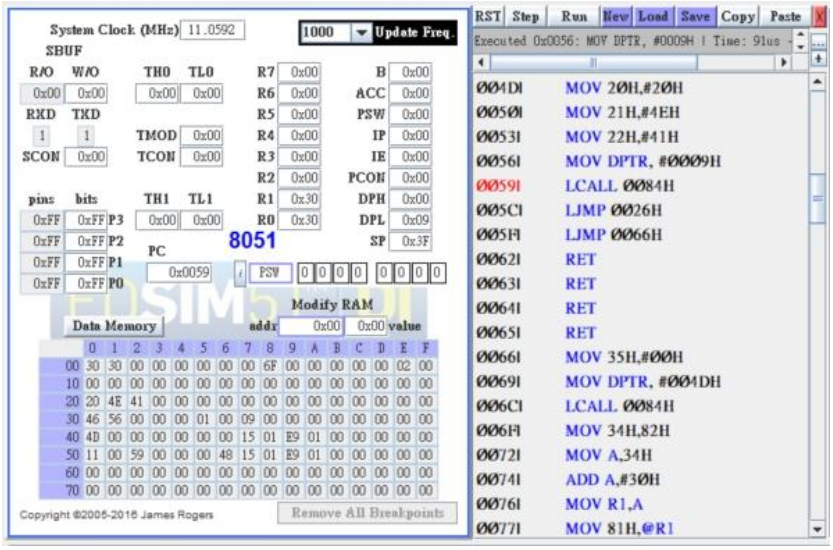
Testcoop.map

```
      Value  Global                           Global Defined In Module
      -----  --------------------------------  ------------------------
C:    00000009  _Producer                       testcoop
C:    00000026  _Consumer                       testcoop
C:    0000004D  _main                           testcoop
C:    0000005F  __sdcc_gsinit_startup           testcoop
C:    00000063  __mcs51_genRAMCLEAR             testcoop
C:    00000064  __mcs51_genXINIT                testcoop
C:    00000065  __mcs51_genXRAMCLEAR            testcoop
C:    00000066  _Bootstrap                      cooperative
C:    00000084  _ThreadCreate                   cooperative
C:    000000F3  _ThreadYield                    cooperative
C:    0000014E  _ThreadExit                     cooperative
```

From testcoop.map, it shows that the address of the ThreadCreate() function is at 00000084, so I set checkpoint at 0084 and then run the Edsim51.

ThreadCreate



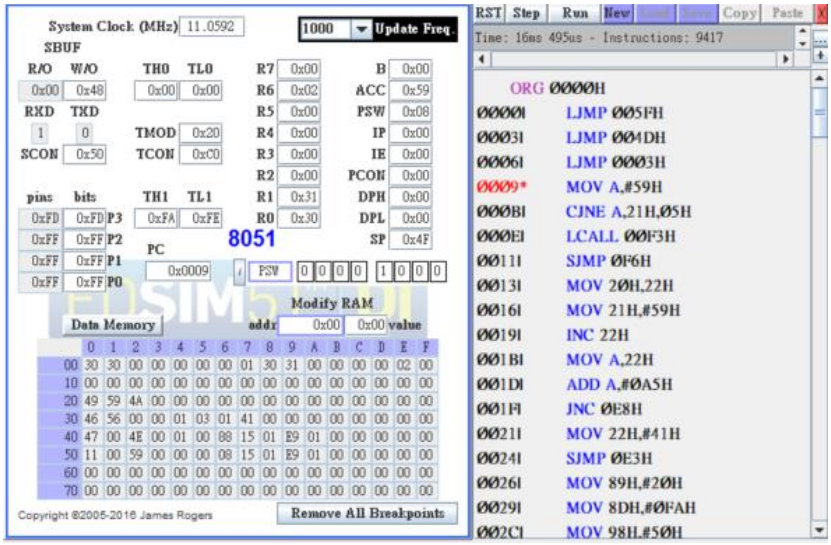


0x40 is changed since it push new data to the stack.

0x40 and 0x41 is the main address which can observe from the testcoop.map. (4D).

0x35 to 0x37 is changed since the bitmap(0x35) and final_id are changed.(0x36, return from ThreadCreate())

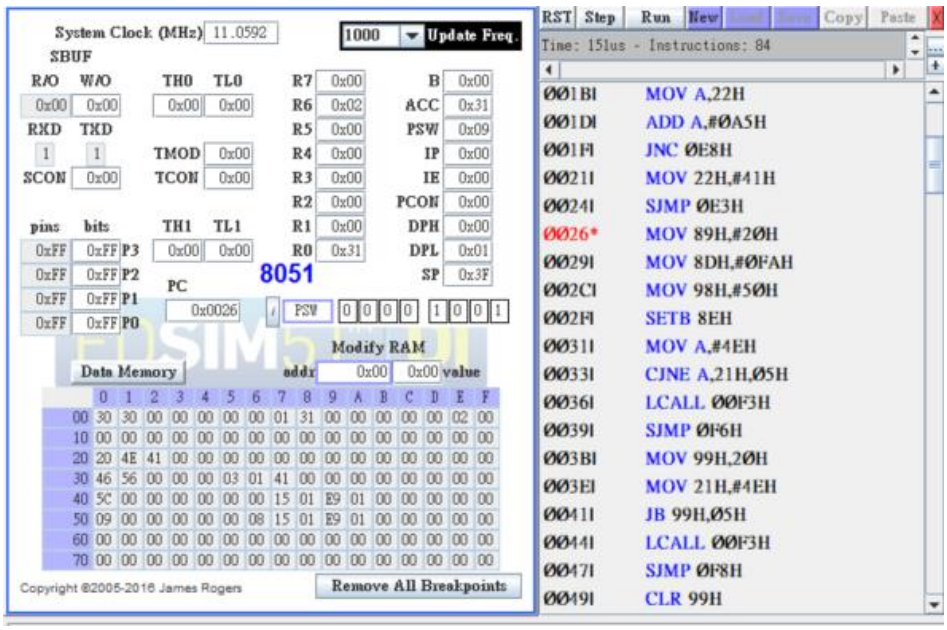
Procuder



From testcoop.map, we can also tell that the address of Producer is at 0009. Set checkpoint to it and run Edsim51.

To know whether the Producer is running, it can tell from the address 0x34, which points to the current_id. (current thread id) The value of it is 01, which means Producer is running.

Consumer



Same as the Producer, we can observe that the address of Consumer is at 0026.

To know whether the Consumer is running, the current_id is 00, which means Consumer is running.

Code Explanations

Testcoop.c

Global Variables

```
__data __at (0x20) char buffer;
__data __at (0x21) char full;
__data __at (0x22) char letter;
```

- Full : checking whether buffer is full
- Letter : Indicate which letter is next

Producer()

```
while (1) {
        /* @@@ [6 pt]
         * wait for the buffer to be available,
         * and then write the new data into the buffer */

        while (full == 'Y')
            ThreadYield();

        buffer = letter;
        full = 'Y';

        letter++;

        if (letter > 'Z'){
            letter = 'A';
        }
}
```

- When buffer is full, yield.
- Letter write into buffer and set full to Yes.
- If letter is over Z, set it back to A

Consumer()

```
TMOD = 0x20;
TH1 = -6;
SCON = 0x50;
TR1 = 1;

while (1) {
        /* @@@ [2 pt] wait for new data from producer
         * @@@ [6 pt] write data to serial port Tx,
         * poll for Tx to finish writing (TI),
         * then clear the flag
         */

        while (full == 'N')
            ThreadYield();

        SBUF = buffer;
        full = 'N';

        while(TI==0)
                ThreadYield();
        TI = 0;


}
```

- Initialize Tx for polling
- When buffer is empty, yield.
- Write buffer to SBUF and set full to No.
- When Tx is not ready, yield.
- Set TI to 0 to keep it from sending.

Cooperative.c

Global Variables

```
__data __at (0x30) char SSP[MAXTHREADS];
__data __at (0x34) char current_id;
__data __at (0x35) char bitmap;
__data __at (0x36) char final_id;
__data __at (0x37) char tmp_SP;
```

- SSP : Save stack pointer

```
#define SAVESTATE \
        __asm \
            PUSH ACC\
            PUSH B\
            PUSH DPL\
            PUSH DPH\
            PUSH PSW\
        __endasm; \
        SSP[current_id] = SP;\
```

```
#define RESTORESTATE \
        SP = SSP[current_id];\
          __asm \
            POP PSW\
            POP DPH\
            POP DPL\
            POP B\
            POP ACC\
          __endasm; \
```

- Push state to the stack, Save and restore will do in opposite way.

ThreadCreate()

```
if (bitmap == 15){
    return -1;
}

tmp_SP = SP;


switch(bitmap){
    case 0:
        final_id = 0;
        bitmap = 1;
        SP = 0x3F;
        break;
    case 1:
        final_id = 1;
        bitmap = 3;
        SP = 0x4F;
        break;
    case 3:
        final_id = 2;
        bitmap = 7;
        SP = 0x5F;
        break;
    case 7:
        final_id = 3;
        bitmap = 15;
        SP = 0x6F;
        break;
}


//PSW = 0;
PSW = final_id << 3;

__asm
    PUSH DPL
    PUSH DPH
    ANL A,#0
    PUSH ACC
    PUSH ACC
    PUSH ACC
    PUSH ACC
    PUSH PSW
__endasm;

SSP[final_id] = SP;
SP = tmp_SP;

return final_id;
```

1. Check bitmap to confirm that there are still threads available.
2. To go back, save stack pointer by tmp_SP.
3. Based on bitmap, set new bitmap, the id to return and Stack Pointer.
4. Set PSW
5. Push DPL, DPH, ACC, B, and PSW.
6. Store current stack pointer to SSP.
7. Set current stack pointer to tmp_SP.
8. Return the new id.

ThreadYield()

```
current_id++;

    if (bitmap == 0b0001)
        if (current_id > 0)
            current_id = 0;
    if (bitmap == 0b0011)
        if (current_id > 1)
            current_id = 0;
    if (bitmap == 0b0111)
        if (current_id > 2)
            current_id = 0;
    if (bitmap == 0b1111)
        if (current_id > 3)
            current_id = 0;

break;
```

Implement Round Robin

- Plus current_id by 1.
- If current id is bigger than the max thread in use, set it back to 0.