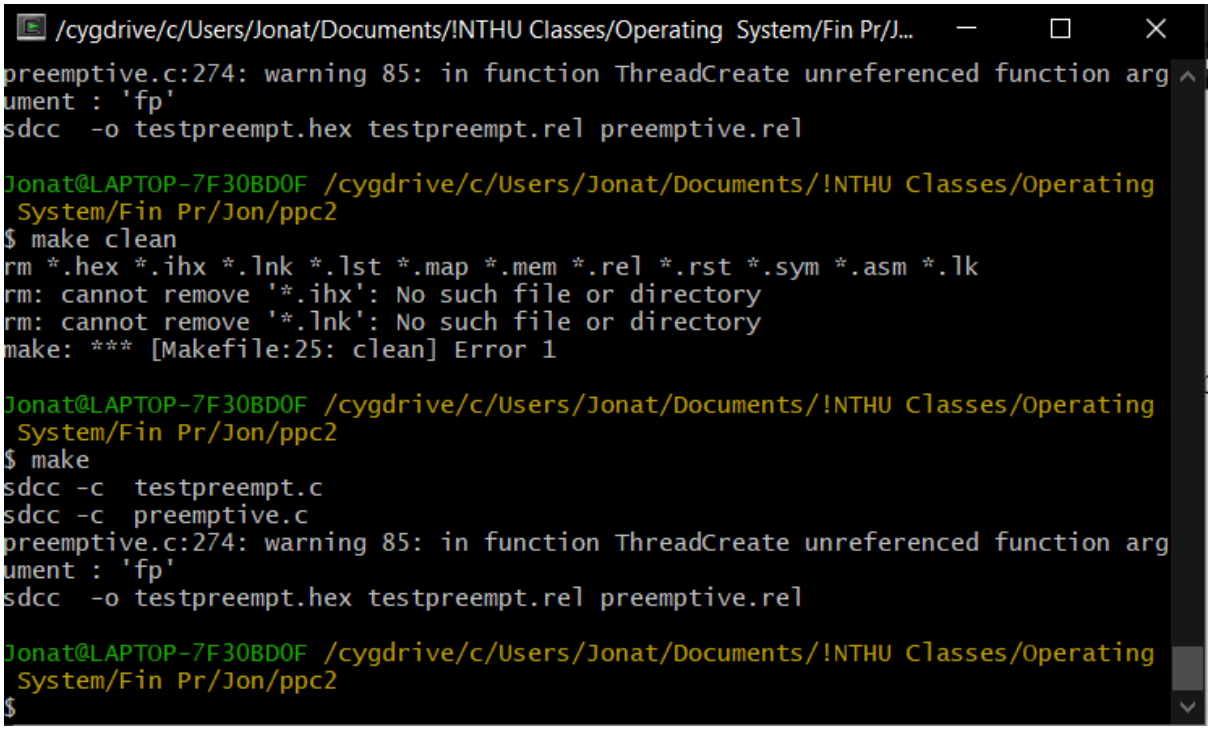
Report Checkpoint 2

Jonathan Sutedjo 鄭安良 111006207

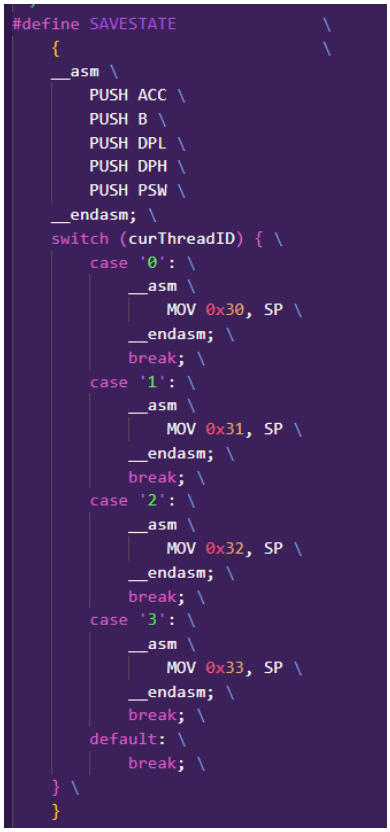


```c
void Producer(void)
{
    /*
     * [TODO]
     * initialize producer data structure, and then enter
     * an infinite loop (does not return)
     */
    __critical{
        SharedBuffer = 'A'-1;
    }
    while (1)
    {
        /* [TODO]
         * wait for the buffer to be available,
         * and then write the new data into the buffer */
        if(Buffer_Availability){
        }
        else{
            __critical{
                Buffer_Availability += 1;
            }
            if(SharedBuffer == 'Z'){
                __critical{
                    SharedBuffer = 'A';
                }
            }
            else{
                __critical{
                    SharedBuffer += 1;
                }
            }
        }
    }
}
```

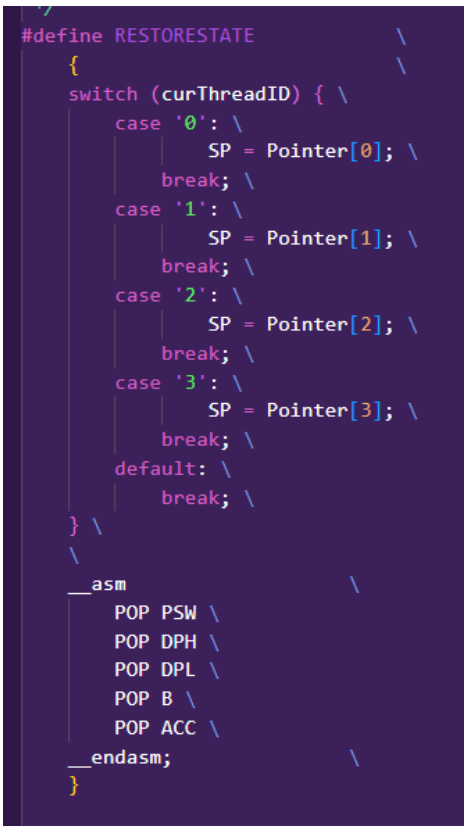
```c
void Consumer(void)
{
    /*
     * [TODO]
     * initialize Tx for polling
     */
    TMOD |= 0x20;
    TH1 = (char)-6;
    SCON = 0x50;
    TR1 = 1;
    TI = 1;

    while (1)
    {
        /*
         * [TODO]
         * wait for new data from producer
         * write data to serial port Tx,
         * poll for Tx to finish writing (TI),
         * then clear the flag
         */
        if(Buffer_Availability){
            while (!TI){
            }
            SBUF = SharedBuffer;
            TI = 0;
            __critical{
                Buffer_Availability -= 1;
            }
        }
    }
}
```

Above are my producer and consumer, they have the same functionality as checkpoint 1 but the changes that I made are the addition of removing the “ThreadYield()” and adding the wrapper “__critical” to some parts of my code. I also made a change to the consumer function where I change the “TMOD = 0x20” to become “TMOD |= 0x20”.

In the producer part, it is responsible for generating the characters in a sequential order from ‘A’ to ‘Z’ and storing it in the shared buffer, meanwhile in the consumer part, it continues to retrieve character from the shared buffer and writes them to the serial port.



```
#define SAVESTATE                \
    {                            \
    __asm \
        PUSH ACC \
        PUSH B \
        PUSH DPL \
        PUSH DPH \
        PUSH PSW \
    __endasm; \
    switch (curThreadID) { \
        case '0': \
            __asm \
                MOV 0x30, SP \
            __endasm; \
            break; \
        case '1': \
            __asm \
                MOV 0x31, SP \
            __endasm; \
            break; \
        case '2': \
            __asm \
                MOV 0x32, SP \
            __endasm; \
            break; \
        case '3': \
            __asm \
                MOV 0x33, SP \
            __endasm; \
            break; \
        default: \
            break; \
    } \
    }
```



```
#define RESTORESTATE             \
    {                            \
    switch (curThreadID) { \
        case '0': \
                SP = Pointer[0]; \
            break; \
        case '1': \
                SP = Pointer[1]; \
            break; \
        case '2': \
                SP = Pointer[2]; \
            break; \
        case '3': \
                SP = Pointer[3]; \
            break; \
        default: \
            break; \
    } \
    \
    __asm                    \
        POP PSW \
        POP DPH \
        POP DPL \
        POP B \
        POP ACC \
    __endasm;                \
    }
```

Above are my savestate and restorestate function where savestate saves the state of the current thread by pushing the key CPU registers (ACC, B, DPL, DPH, and PSW) onto the stack and storing the stack pointer into the array indexed by the current thread ID, and where restorestate restores the state of the current thread by loading the saved stack pointer from the array and popping the key CPU registers (PSW, DPL, DPH, B, and ACC) from the stack.

```
void Bootstrap(void)
{
    /*
     * [TODO]
     * initialize data structures for threads (e.g., mask)
     *
     * optional: move the stack pointer to some known location
     * only during bootstrapping. by default, SP is 0x07.
     *
     * [TODO]
     *     create a thread for main; be sure current thread is
     *     set to this thread ID, and restore its context,
     *     so that it starts running main().
     */

    ValidBitMap = 0b0000;
    Pointer[0] = 0x3F;
    Pointer[1] = 0x4F;
    Pointer[2] = 0x5F;
    Pointer[3] = 0x6F;
    TMOD = 0; //timer zero
    IE = 0x82;
    TR0 = 1;
    curThreadID = ThreadCreate(main);
    RESTORESTATE;
}
```

For my bootstrap, I initialized the TMOD, IE, and TR0 here and make the TMOD = 0 while the TR0 and IE stays the same. The other part of the code here, I do not make any change and still the same with checkpoint 1. The change is only the “TMOD = 0”.

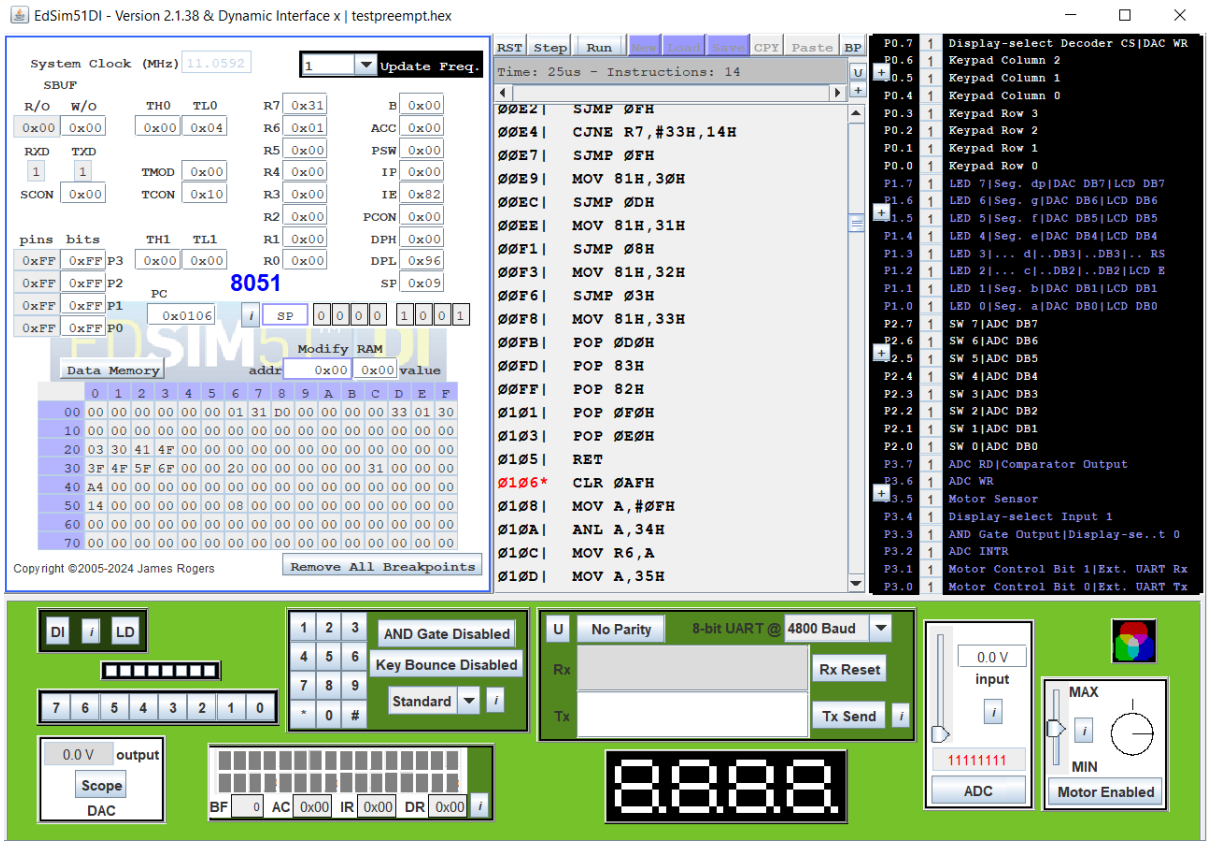
```
void myTimer0Handler(void){
    EA = 0;
    SAVESTATE;
    do{
        curThreadID = (curThreadID == '3') ? '0' : curThreadID + 1;

        switch (curThreadID) {
            case '0':
                if ((ValidBitMap & 0b0001) == 0b0001) break;
                continue;
            case '1':
                if ((ValidBitMap & 0b0010) == 0b0010) break;
                continue;
            case '2':
                if ((ValidBitMap & 0b0100) == 0b0100) break;
                continue;
            case '3':
                if ((ValidBitMap & 0b1000) == 0b1000) break;
                continue;
        }
        break;
      } while (1);
      RESTORESTATE;

      EA = 1;
      __asm
          RETI
      __endasm;
}
```

This is my myTimer0Handler where I implement something like ThreadYield() that I removed previously in some parts of my code for this part 2. The difference between this function with the ThreadYield is where I use RETI instead of RET.

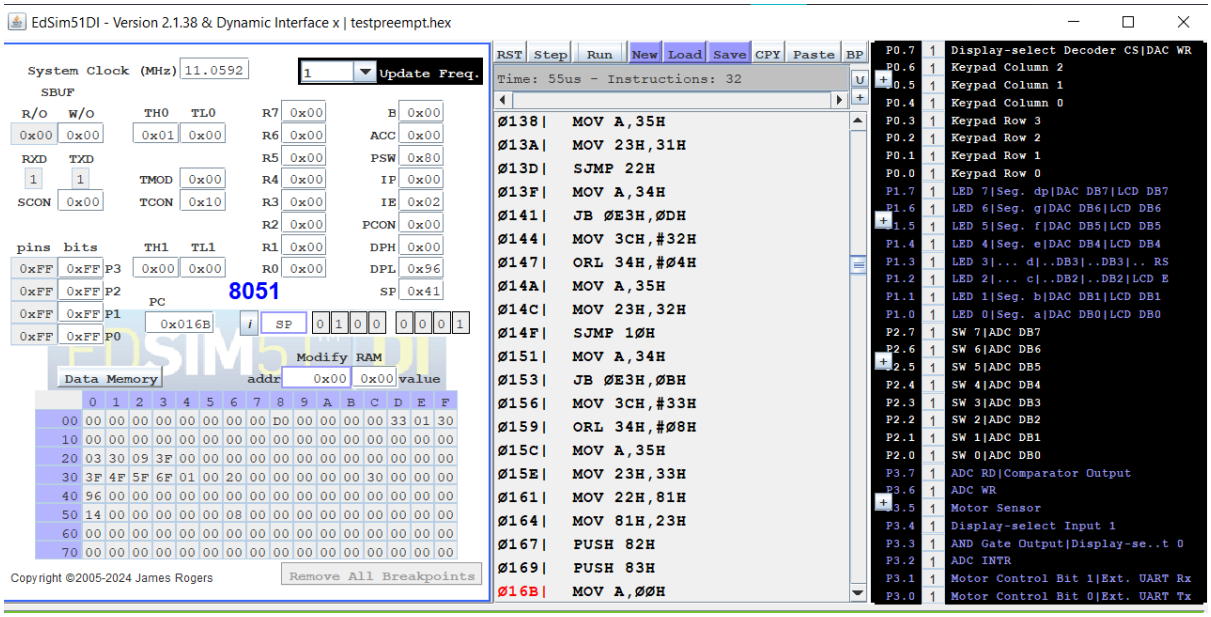
Other part of my code are still the same with previous, the other changes I made is in the makefile because of changing the file name. Other than that the header also as I made changes to the file name.

```
      Value  Global                                  Global Defined In Module
      -----  ----------------------------------      ------------------------
C:    00000014  _Producer                              testpreempt
C:    00000065  _Consumer                              testpreempt
C:    00000096  _main                                  testpreempt
C:    000000A7  __sdcc_gsinit_startup                  testpreempt
C:    000000AB  __mcs51_genRAMCLEAR                    testpreempt
C:    000000AC  __mcs51_genXINIT                       testpreempt
C:    000000AD  __mcs51_genXRAMCLEAR                   testpreempt
C:    000000AE  _timer0_ISR                            testpreempt
C:    000000B2  _Bootstrap                             preemptive
C:    00000106  _ThreadCreate                          preemptive
C:    000001BA  _ThreadYield                           preemptive
C:    0000029A  _ThreadExit                            preemptive
C:    00000381  _myTimer0Handler                       preemptive
```

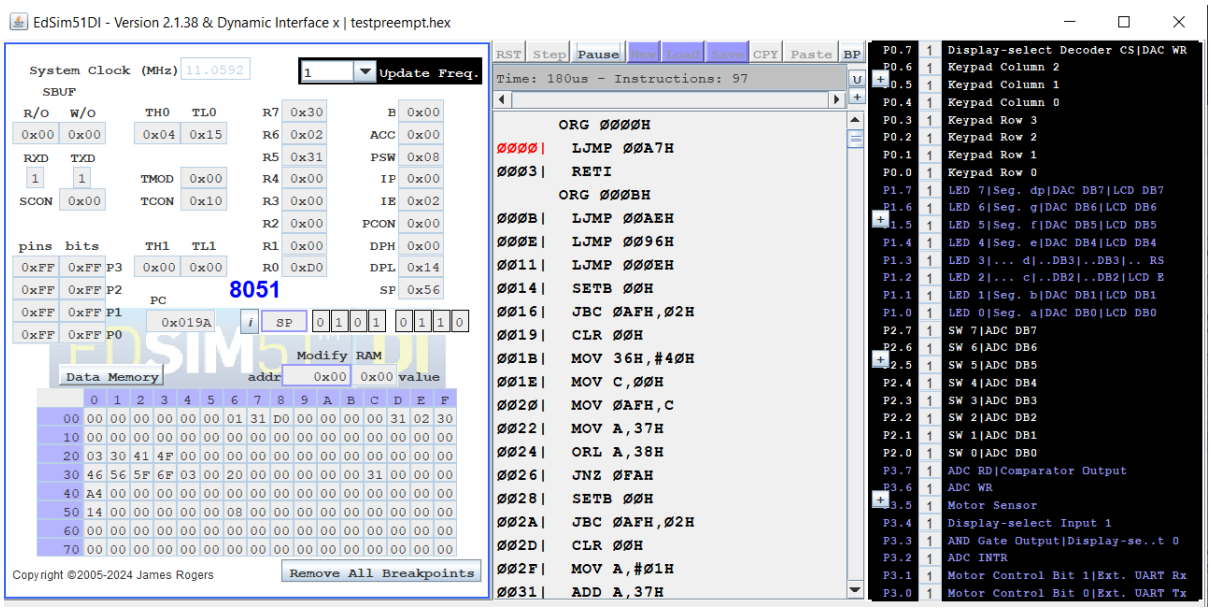
This is some content of my testpreempt.map file where it shows addresses.



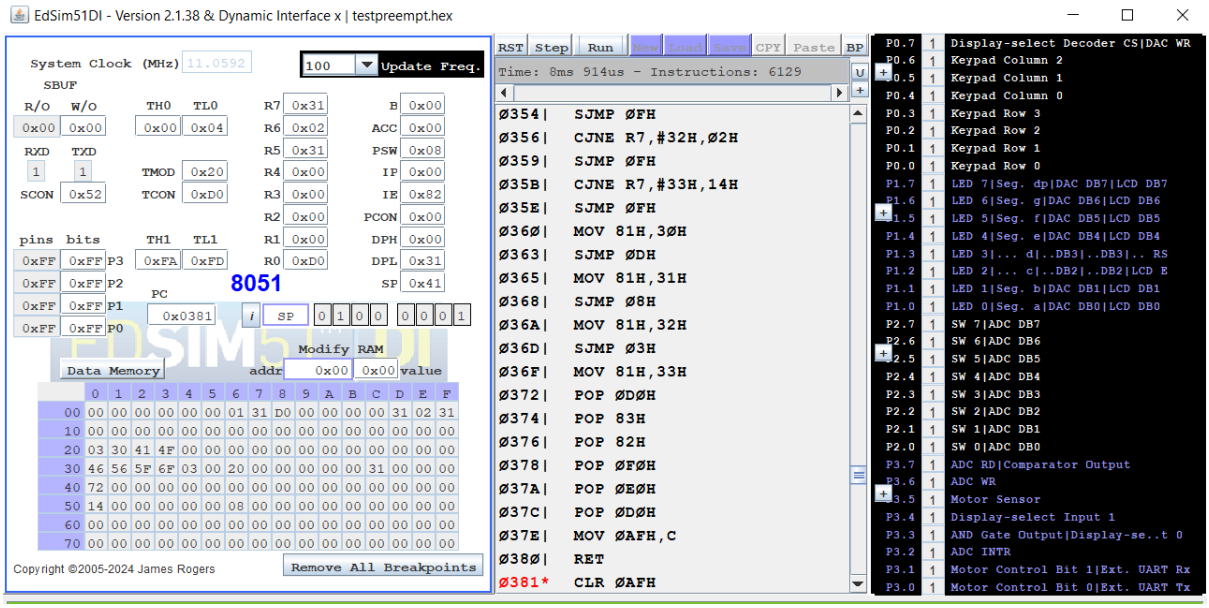
Here, this is moments before ThreadCreate(main) was called, I can know that because the PC is 0x0106 and it is the same with the data in the map file where the ThreadCreate first starts. I can also see that the SP at that time point is 0x09 which is different from the starting 0x07.



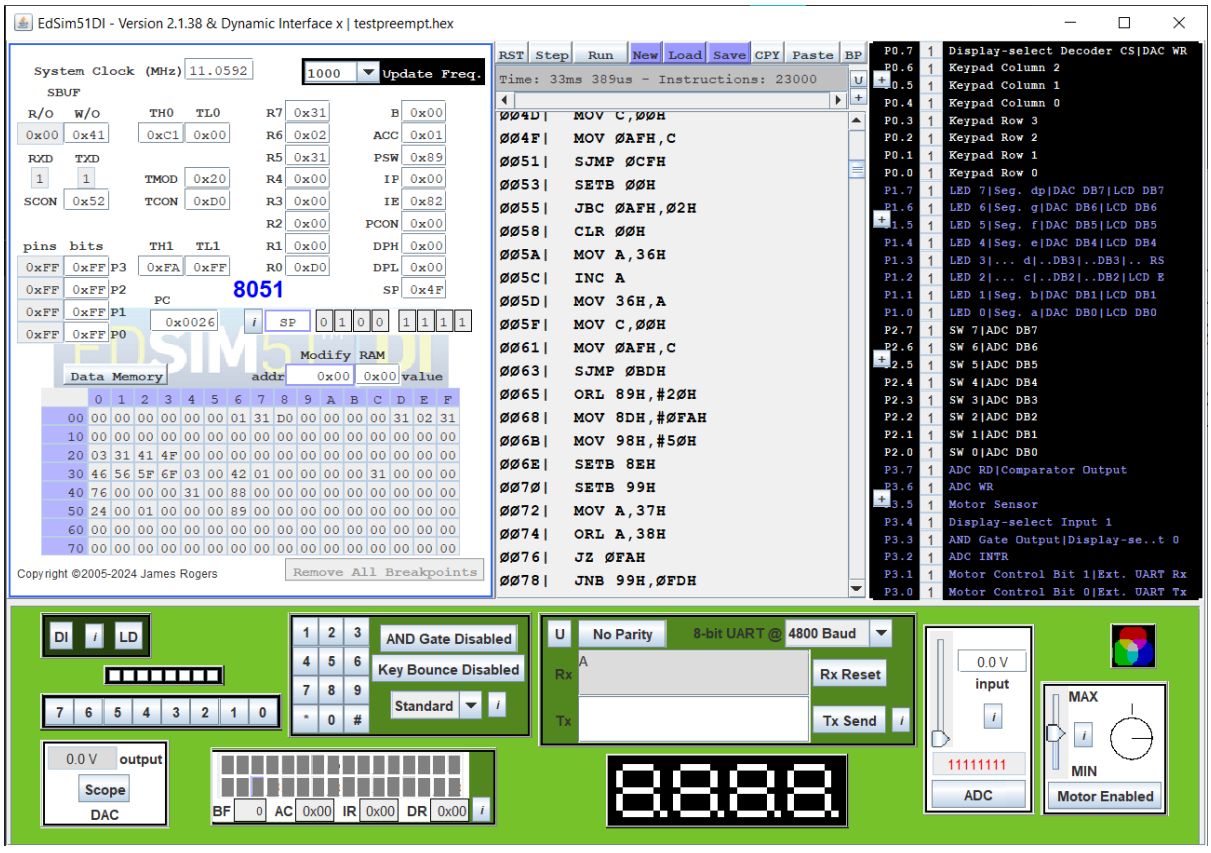
Here, this is before the second ThreadCreate was called, the SP is now in the range of 40-4F which means that we are now in the main thread.



This is the time before the MyTimer0Handler is called because here the SP is in the range of 50-5F which means that the thread that is running is the producer.



This is the time before the MyTimer0Handler is called because we can see the range of the SP is 40-4F right now and that means the consumer is now running.



Above you can see that it finally printed out “A” that means the consumer works and it runs.

We can know that the interrupt will trigger on a regular basis because when the Timer0 overflows, interrupt happens and reset the timer back to 0.