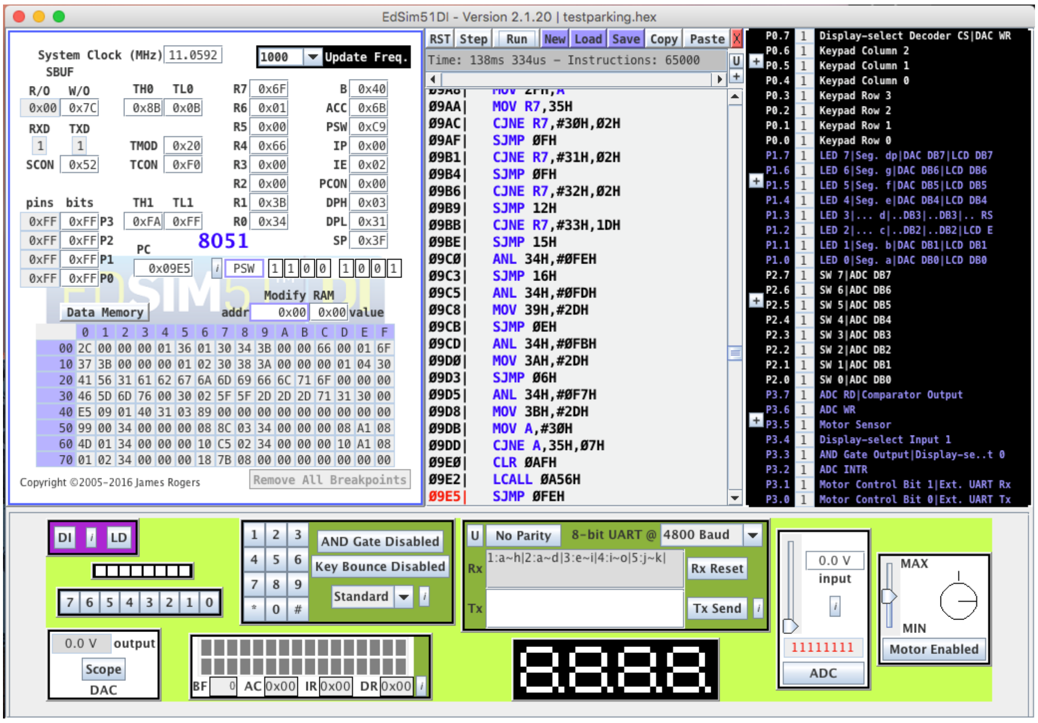
Programming Project Checkpoint 5

105061171 陳冠弘

Screenshot:



Explanation:

In this project, I use one thread to be a “car creator” and other three threads to be cars, running the parking lot exmple. At the Rx, we see that for each car, it has a period of time such as a~h which indicated that this car occupies a parking lot from time a to time h.

Screenshot:

```
void delay(unsigned char n) {
        /* set up a delay for current thread */
        EA = 0;
        switch (currentThreadID){
            case '1':
                delays[0] = n;
                break;
            case '2':
                delays[1] = n;
                break;
            case '3':
                delays[2] = n;
                break;
            default:
                break;
        }
        EA = 1;
}
```

Explanation:

In delay(n) function, given a length of time n, it sets the array delays[i] to be n. The array stores delay times for each car, and the index is determined by the current thread ID.

Screenshot:

```
unsigned char now(void) {
        /* return the current time stamp */
        return Time;
}
```

Explanation:

In now() function, it returns the current time which is just the variable "Time". This variable is maintained at each thread switching (preemption and yield), each time we increment it by one unit.

- My timer-0 ISR supports delay(n) function by decrementing delay times if any, and checking if some thread's delay time is zero then choose it as the next thread. And it has nothing to do with now() function.
- If multiple threads call delay(n), it will select them in the increasing order (thread 1 first and so on) in a time unit, so it will not miss any delay request.
- If multiple threads finish their delay, it will select them also in the increasing order (thread 1 first and so on) in a time unit.

Screenshot:

```c
void ThreadExit(void) {
        /* terminate a thread and recycle its resource */
        numOfThreads --;
        switch (currentThreadID) {
            case '0':
                __asm
                    ANL 0x34, #0xFE
                __endasm;
                break;
            case '1':
                __asm
                    ANL 0x34, #0xFD
                __endasm;
                delays[0] = '-';
                break;
            case '2':
                __asm
                    ANL 0x34, #0xFB
                __endasm;
                delays[1] = '-';
                break;
            case '3':
                __asm
                    ANL 0x34, #0xF7
                __endasm;
                delays[2] = '-';
                break;
        }
        if (currentThreadID == '0') {
            // main component exits, print out parking information
            EA = 0;
            PrintParkingResult();
            while (1) {}  // infinite loop
        } else {
            // thread exits, switch to main thread to create a new one if needed
            if (thrd_4 == 'x') thrd_4 = (currentThreadID == '1') ? '0' : '1';
            else if (thrd_5 == 'x') {
                if (currentThreadID == '1') thrd_5 = '0';
                else if (currentThreadID == '2') thrd_5 = '1';
                else if (currentThreadID == '3') thrd_5 = '2';
            }
            currentThreadID = '0';
        }
        RESTORESTATE;
}
```

Explanation:

When a thread is terminated, we update the thread map and delay time of it, then we switch to thread 0 (the car creator) to create a new car if needed. In addition, because the first three cars use the remaining three threads, but the fourth and fifth car may use different thread depending on the testcase, so we use two variables to track which thread is terminated and used by car 4 and 5, then use these variables to know which thread we need to switch to if car 4 or 5 is selected.

If thread 0 is terminated, it means that the parking lot example is finished, so we print out the information for each car's entering and leaving time, then jump into an infinite loop instead of returning to nowhere.

Thread creation is almost same as previous checkpoints, we find an available by checking thread map, then we initialize parameters such as PSW, stack pointer and delay time of it, finally we return the ID of this created thread.

Screenshot:

```
void main(void) {
        /* main component for parking lot example */
        SemaphoreCreate(&mutex, 2);  // semaphore for spots

        ThreadCreate(Parking1);
        ThreadCreate(Parking2);
        ThreadCreate(Parking3);

        TMOD |= 0x20;
        TH1 = (char)-6;
        SCON = 0x50;
        TR1 = 1;
        TI = 1;

        ThreadYield();  // start parking

        while (numOfThreads >= MAXTHREADS) {ThreadYield();}
        ThreadCreate(Parking4);  // have available thread, create a new one for car 4

        while (numOfThreads >= MAXTHREADS) {ThreadYield();}
        ThreadCreate(Parking5);  // have available thread, create a new one for car 5

        while (numOfThreads != 1) {ThreadYield();}

        ThreadExit();  // all parking finished, exit this example
}
```

Explanation:

As mentioned above, thread 0 (main function) is used to create cars, each spawn is controlled in this function.

In main function, we set semaphore to 2 as the number of parking slots, then we create cars in thread 1, 2, 3. Then we yield to another thread to start parking lot example. While running this example, any thread switching will not choose thread 0 until any thread is terminated. Then main function will create car 4 and 5 in available threads.

Finally, when the parking lot example is finished, main function terminates, prints information, enters infinite loop.

Screenshot:

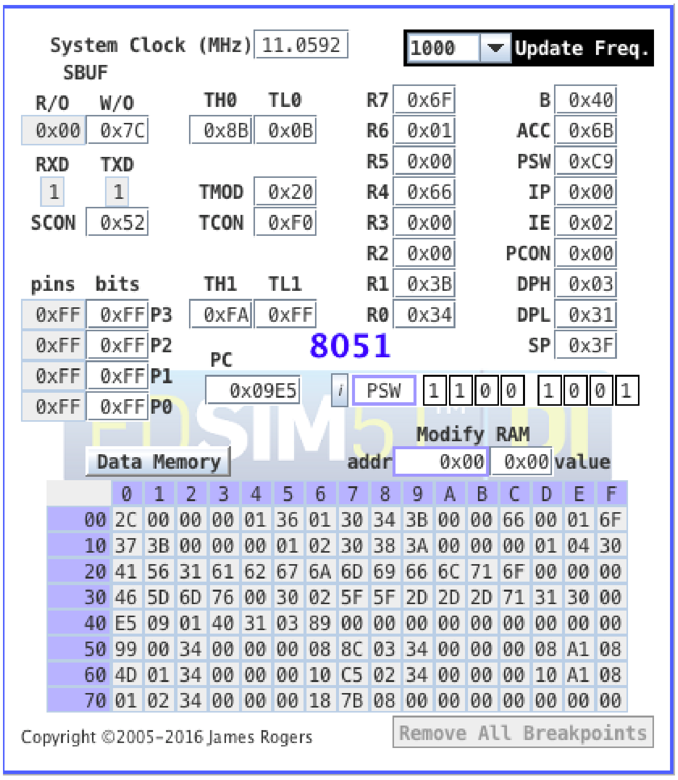
```c
void Parking1(void) {
        /* car 1 for parking lot example */
        while (1) {
                while (_compare(&delays[0], '0') == 0) {ThreadYield();}
                for (i=0; i<2; i++) {
                        if (_compare(&spots[i], '1') == 1) {
                                // already in parking lot, leave now
                                spots[i] = '_';
                                LeaveTimes[0] = now();
                                SemaphoreSignal(mutex);
                                ThreadExit();
                        }
                }
                for (i=0; i<2; i++) {
                        if (_compare(&spots[i], '_') == 1) {
                                // find out a parking lot, enter now
                                SemaphoreWait(mutex);
                                spots[i] = '1';
                                delay('8');
                                EnterTimes[0] = now();
                                break;
                        }
                        if (i == 1) delay('2');
                }
        }
}
```

Explanation:

In each car, once it runs, it searches for an available spot. If it successfully finds a spot, it occupies by setting the spot to its thread ID, then it delays by an amount of time depending on testcase. Finally, it will store the current time as its entering time.

When it finds a spot is occupied by itself, and of course its delay time is zero (otherwise it will not be selected to run), it is time for it to leave, this car resets the spot, stores the current time as its leaving time, terminates itself.

Screenshot:



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	2C	00	00	00	01	36	01	30	34	3B	00	00	66	00	01	6F
10	37	3B	00	00	00	01	02	30	38	3A	00	00	00	01	04	30
20	41	56	31	61	62	67	6A	6D	69	66	6C	71	6F	00	00	00
30	46	5D	6D	76	00	30	02	5F	5F	2D	2D	2D	71	31	30	00
40	E5	09	01	40	31	03	89	00	00	00	00	00	00	00	00	00
50	99	00	34	00	00	00	08	8C	03	34	00	00	00	08	A1	08
60	4D	01	34	00	00	00	10	C5	02	34	00	00	00	10	A1	08
70	01	02	34	00	00	00	18	7B	08	00	00	00	00	00	00	00

```
__data __at (0x20) int oldSP;
__data __at (0x21) int createdThreadSP;
__data __at (0x22) ThreadID createdThreadID;

__data __at (0x23) char EnterTimes[5];
__data __at (0x28) char LeaveTimes[5];

__data __at (0x2D) char i;
__data __at (0x2E) int numOfThreads;

__data __at (0x30) int th0_SP;
__data __at (0x31) int th1_SP;
__data __at (0x32) int th2_SP;
__data __at (0x33) int th3_SP;
__data __at (0x34) int threadBitMap;
__data __at (0x35) ThreadID currentThreadID;

__data __at (0x36) char mutex;
__data __at (0x37) char spots[2];
__data __at (0x39) char delays[3];
__data __at (0x3C) char Time;
__data __at (0x3D) char thrd_4;
__data __at (0x3E) char thrd_5;
```

Explanation:

At the end of parking lot example, we can see that thread map is reset to none, number of thread is reset to zero, each time stamp is recorded.