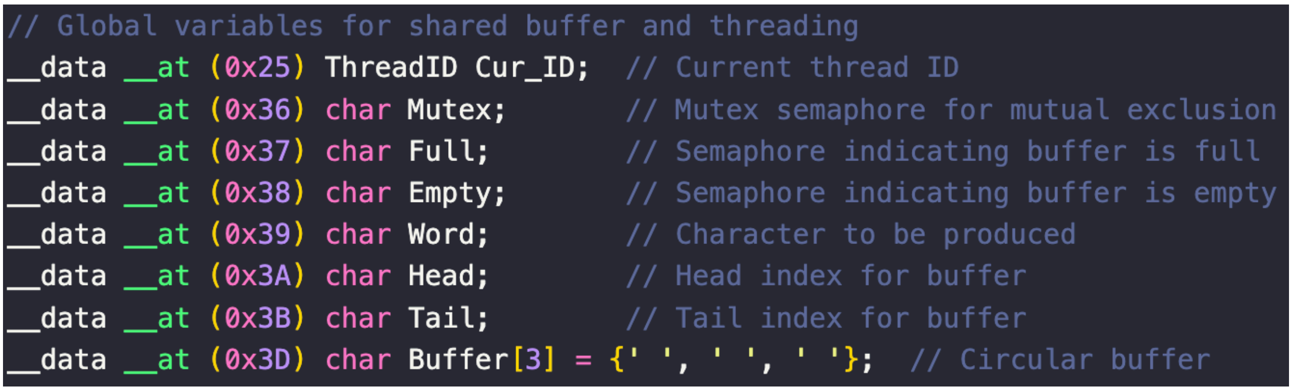
PPC3 Report 110062120 高小榛

1. Code explanation

(1) testpreempt.c

Different with the previous checkpoint, I add a function SemaphoreCreate. This function is to create a semaphore with initial value n.

First, I create global variable Cur_ID, Mutex, Full, Empty, Word, Head, Tail and Buffer. The meaning of these variable is define in this picture.



```
// Global variables for shared buffer and threading
__data __at (0x25) ThreadID Cur_ID;  // Current thread ID
__data __at (0x36) char Mutex;       // Mutex semaphore for mutual exclusion
__data __at (0x37) char Full;        // Semaphore indicating buffer is full
__data __at (0x38) char Empty;       // Semaphore indicating buffer is empty
__data __at (0x39) char Word;        // Character to be produced
__data __at (0x3A) char Head;        // Head index for buffer
__data __at (0x3B) char Tail;        // Tail index for buffer
__data __at (0x3D) char Buffer[3] = {' ', ' ', ' '};  // Circular buffer
```

In the Producer function, I assign first word produce with 'A', then in the while loop, I use SemaphoreWait and SemaphoreSignal to design. First wait Empty until buffer has empty space, then we can enter critical section with wait for Mutex. In the critical section, first add new character to buffer, update tail index then update character. Finally leave the critical section with signal mutex, then signal buffer is full.

In Consumer function, first I initialize Tx for polling using the information in class. , change the TMOD to TMOD |= 0x20. Then we should wait for new data from producer, write data to serial port Tx, poll for Tx to finish writing (TI), then finally clear the flag. So first we should wait for data in buffer with wait Full, then wait for mutex to enter critical section. In the critical section, we can do thing that mention above. Wait for Tx to be ready, send character from buffer, clear Tx interrupt flag then circular increment of head. Finally exit critical section then signal buffer is empty.

In main function, I create 3 semaphore Mutex, Full and Empty. Then initialize head index and tail index. Use ThreadCreate function to create producer thread and store with cur_id. Finally, use assembly to set up stack pointer, call consumer function.

Additionally, I add timer0_ISR to let ISR call my routine timer.

(2) preemptive.c

First, I create the static global SP0 ~ SP3, Map, Cur_ID, SP_old, SP_new and New_ID. Then I define 2 macro save state and restore state in a similar way. In savestate we should saving the current thread context. So first push ACC, B register, Data pointer registers (DPL, DPH), PSW with assembly. Then save SP into the saved Stack Pointers array with if-else block to check which place we should store the SP into it. In restorestate, I first assign SP to the saved SP from the saved stack pointer array, then pop the registers PSW, data pointer registers, B reg, and ACC.

In bootstrap function, I clear thread bitmap indicating no threads are running and initialize stack pointer. I initialize timer and interrupts for preemption with set TMOD = 0, IE = 0x82 to enable timer 0 interrupt, and set TR0 = 1 to start running timer0. Then I create a thread for main and store with Cur_ID and restore the content to run main function.

In myTimer0Handler function, first I use EA=0 to disable interrupts, then save the current thread state. Next I cycle through thread IDs to find the next valid thread and break out of the loop if a valid thread is found. Finally restore the state, re-enable interrupts then return.

In ThreadCreate function, first disable interrupt, check to see we have not reached the max #threads and return -1 if no available thread ID. Then initialize a New_ID, run a if-else block to check each thread ID (0 to 3) to find an available one. For each thread ID, perform bitwise AND with Map and check if the result is 0. If 0, it means the thread ID is available for use. If Thread I is available: first set New_ID to ‘i’, then update Map to indicate Thread I is now in use, finally set SP_new to the starting stack location for Thread i. Then save the current SP in SP_old, set SP to the new thread's starting stack location. Then push the return address (fp) onto the stack for the new thread. Then initialize the registers to 0 for the new thread by set A to 0 then push into each register. Then set up the PSW for the new thread and save its SP, and PSW setup depends on the thread ID to use the correct register bank. Finally restore the original SP from SP_old, re-enable interrupt, and return new_ID.

In ThreadYield function, I use Round-Robin to find the next thread to run. First cycle through thread IDs and select the next valid thread, then break out of the loop if a valid thread is found, finally restore state.

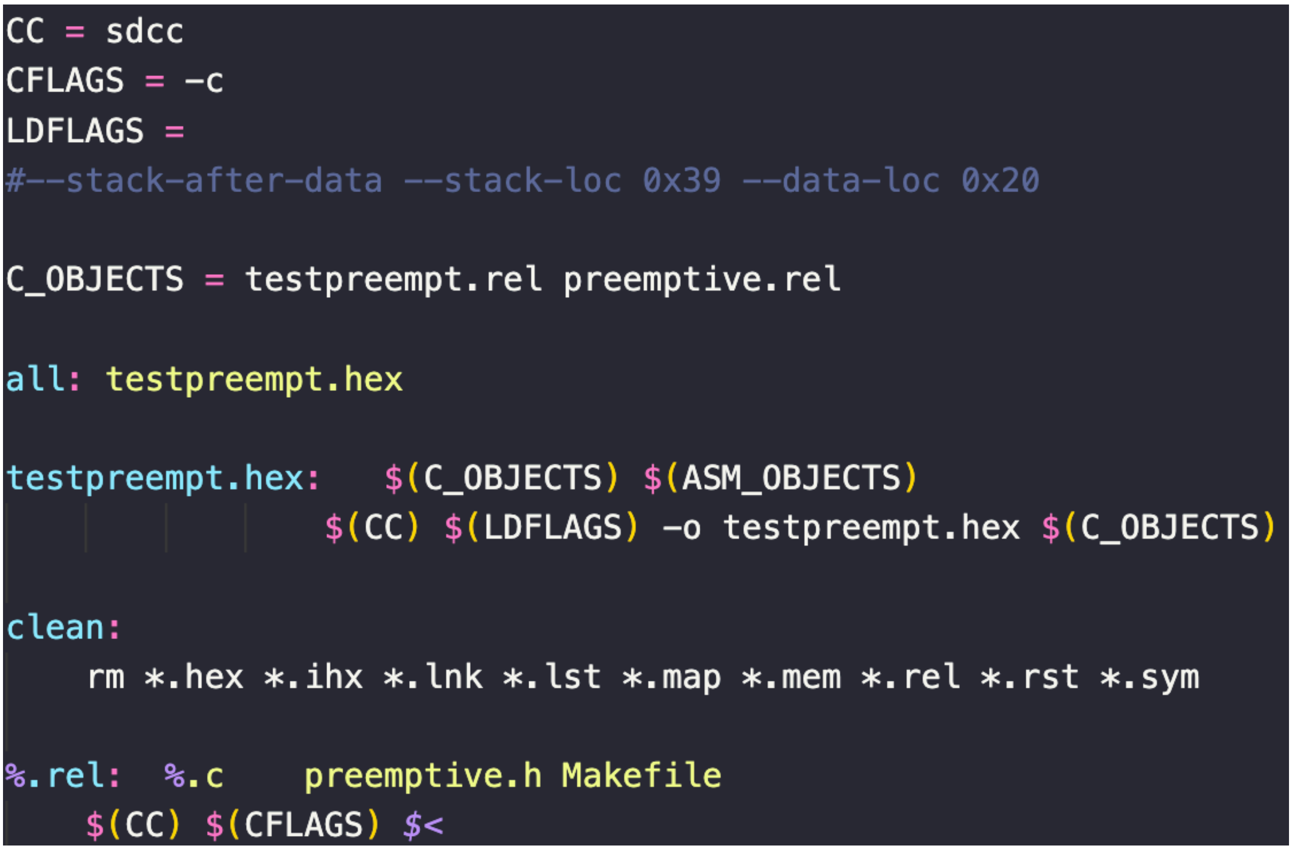
In ThreadExit function, I clear the current thread from the thread bitmap and set up for context switch, then modify the thread bitmap to indicate the current thread is no longer valid. After dealing with Map, also cycle through thread IDs and select the next valid thread, then break out of the loop if a valid thread is found, finally restore state.

(3) preemptive.h

In preemptive.h, I add some definition according to the document. First, CNAME and LABELNAME is to concatenating symbols, it can add _ in front and $ in tail. Then I define SemaphoreSignal is to signal a semaphore. SemaphoreWaitBody do lots of things. First check semaphore value, decoded wait if it is 0, finally decrement semaphore. Last I define SemaphoreWiat, it will use SemaphoreWaitBody to wait a semaphore.

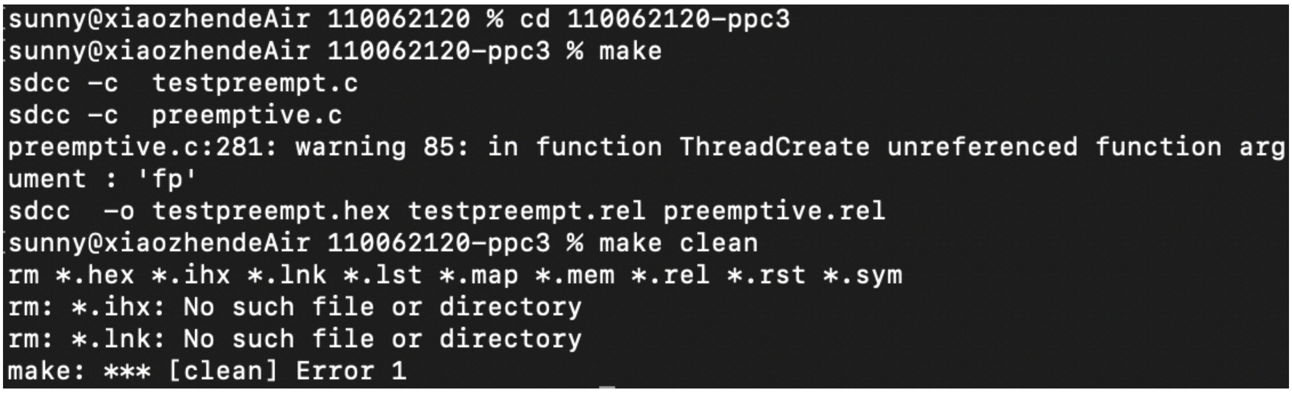
2. Typescript

(1) Makefile



```
CC = sdcc
CFLAGS = -c
LDFLAGS =
#--stack-after-data --stack-loc 0x39 --data-loc 0x20

C_OBJECTS = testpreempt.rel preemptive.rel

all: testpreempt.hex

testpreempt.hex:    $(C_OBJECTS) $(ASM_OBJECTS)
                $(CC) $(LDFLAGS) -o testpreempt.hex $(C_OBJECTS)

clean:
    rm *.hex *.ihx *.lnk *.lst *.map *.mem *.rel *.rst *.sym

%.rel:  %.c    preemptive.h Makefile
    $(CC) $(CFLAGS) $<
```

(2) Screenshot

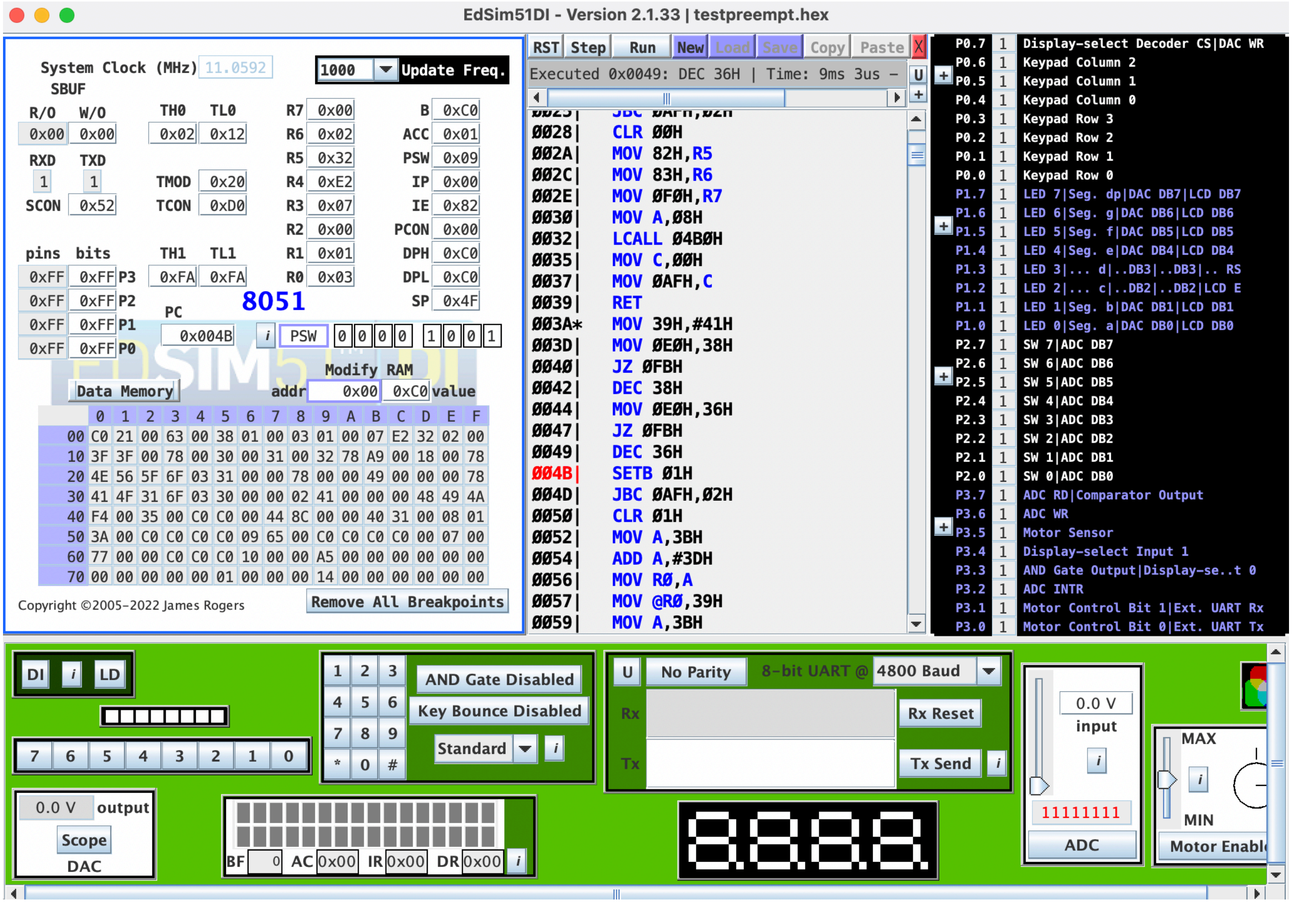


```
sunny@xiaozhendeAir 110062120 % cd 110062120-ppc3
sunny@xiaozhendeAir 110062120-ppc3 % make
sdcc -c  testpreempt.c
sdcc -c  preemptive.c
preemptive.c:281: warning 85: in function ThreadCreate unreferenced function arg
ument : 'fp'
sdcc  -o testpreempt.hex testpreempt.rel preemptive.rel
sunny@xiaozhendeAir 110062120-ppc3 % 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
```

2. Screenshots and explanation

(1) Producer running

Producer is at 003A according to map.



EdSim51DI - Version 2.1.33 | testpreempt.hex

```
System Clock (MHz) 11.0592        1000  Update Freq.
SBUF
R/O   W/O     TH0   TL0     R7 0x00      B    0xC0
0x00  0x00    0x02  0x12    R6 0x02      ACC  0x01
                            R5 0x32      PSW  0x09
RXD   TXD     TMOD  0x20    R4 0xE2      IP   0x00
1     1       TCON  0xD0    R3 0x07      IE   0x82
SCON  0x52                  R2 0x00      PCON 0x00
pins  bits    TH1   TL1     R1 0x01      DPH  0xC0
0xFF  0xFF P3 0xFA  0xFA    R0 0x03      DPL  0xC0
0xFF  0xFF P2                            SP   0x4F
0xFF  0xFF P1  PC 0x004B   PSW 0 0 0 0  1 0 0 1
0xFF  0xFF P0
                    Modify RAM
Data Memory         addr 0x00   0xC0 value

     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
00  C0 21 00 63 00 38 01 00 03 01 00 07 E2 32 02 00
10  3F 3F 00 78 00 30 00 31 00 32 78 A9 00 18 00 78
20  4E 56 5F 6F 03 31 00 00 78 00 00 49 00 00 00 78
30  41 4F 31 6F 03 30 00 00 02 41 00 00 00 48 49 4A
40  F4 00 35 00 C0 C0 00 44 8C 00 00 40 31 00 08 01
50  3A 00 C0 C0 C0 C0 09 65 00 C0 C0 C0 C0 00 07 00
60  77 00 00 C0 C0 C0 10 00 00 A5 00 00 00 00 00 00
70  00 00 00 00 00 01 00 00 00 14 00 00 00 00 00 00
```

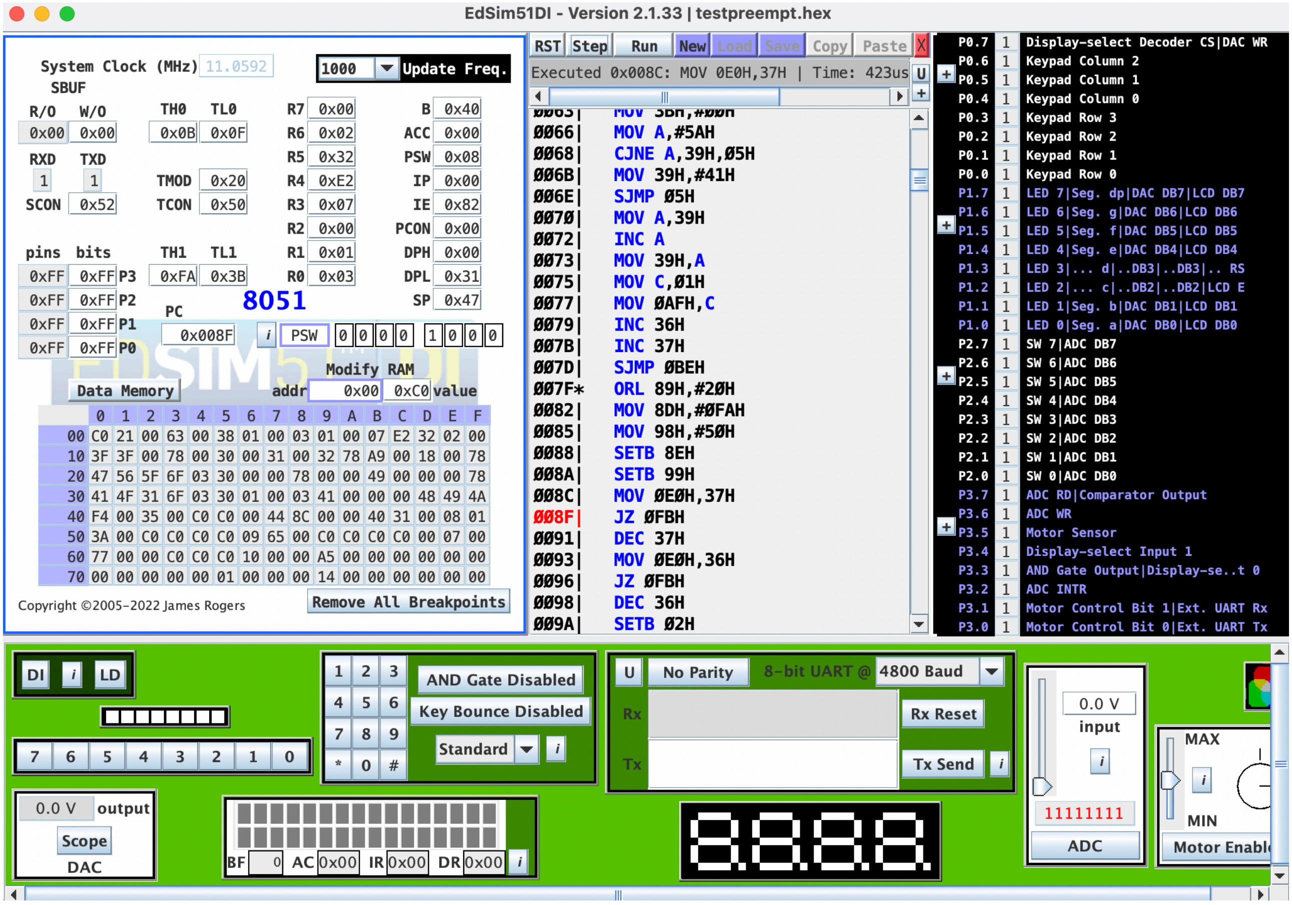
Copyright ©2005-2022 James Rogers

```
Executed 0x0049: DEC 36H | Time: 9ms 3us –
ØØ28|   CLR ØØH
ØØ2A|   MOV 82H,R5
ØØ2C|   MOV 83H,R6
ØØ2E|   MOV ØFØH,R7
ØØ3Ø|   MOV A,Ø8H
ØØ32|   LCALL Ø4BØH
ØØ35|   MOV C,ØØH
ØØ37|   MOV ØAFH,C
ØØ39|   RET
ØØ3A*   MOV 39H,#41H
ØØ3D|   MOV ØEØH,38H
ØØ4Ø|   JZ ØFBH
ØØ42|   DEC 38H
ØØ44|   MOV ØEØH,36H
ØØ47|   JZ ØFBH
ØØ49|   DEC 36H
ØØ4B|   SETB Ø1H
ØØ4D|   JBC ØAFH,Ø2H
ØØ5Ø|   CLR Ø1H
ØØ52|   MOV A,3BH
ØØ54|   ADD A,#3DH
ØØ56|   MOV RØ,A
ØØ57|   MOV @RØ,39H
ØØ59|   MOV A,3BH
```

```
P0.7 1 Display-select Decoder CS|DAC WR
P0.6 1 Keypad Column 2
P0.5 1 Keypad Column 1
P0.4 1 Keypad Column 0
P0.3 1 Keypad Row 3
P0.2 1 Keypad Row 2
P0.1 1 Keypad Row 1
P0.0 1 Keypad Row 0
P1.7 1 LED 7|Seg. dp|DAC DB7|LCD DB7
P1.6 1 LED 6|Seg. g|DAC DB6|LCD DB6
P1.5 1 LED 5|Seg. f|DAC DB5|LCD DB5
P1.4 1 LED 4|Seg. e|DAC DB4|LCD DB4
P1.3 1 LED 3|... d|..DB3|..DB3|.. RS
P1.2 1 LED 2|... c|..DB2|..DB2|LCD E
P1.1 1 LED 1|Seg. b|DAC DB1|LCD DB1
P1.0 1 LED 0|Seg. a|DAC DB0|LCD DB0
P2.7 1 SW 7|ADC DB7
P2.6 1 SW 6|ADC DB6
P2.5 1 SW 5|ADC DB5
P2.4 1 SW 4|ADC DB4
P2.3 1 SW 3|ADC DB3
P2.2 1 SW 2|ADC DB2
P2.1 1 SW 1|ADC DB1
P2.0 1 SW 0|ADC DB0
P3.7 1 ADC RD|Comparator Output
P3.6 1 ADC WR
P3.5 1 Motor Sensor
P3.4 1 Display-select Input 1
P3.3 1 AND Gate Output|Display-se..t 0
P3.2 1 ADC INTR
P3.1 1 Motor Control Bit 1|Ext. UART Rx
P3.0 1 Motor Control Bit 0|Ext. UART Tx
```

(2) Consumer running

Consumer is at 007F according to map.



EdSim51DI - Version 2.1.33 | testpreempt.hex

```
System Clock (MHz) 11.0592        1000  Update Freq.
SBUF
R/O   W/O     TH0   TL0     R7 0x00      B    0x40
0x00  0x00    0x0B  0x0F    R6 0x02      ACC  0x00
                            R5 0x32      PSW  0x08
RXD   TXD     TMOD  0x20    R4 0xE2      IP   0x00
1     1       TCON  0x50    R3 0x07      IE   0x82
SCON  0x52                  R2 0x00      PCON 0x00
pins  bits    TH1   TL1     R1 0x01      DPH  0x00
0xFF  0xFF P3 0xFA  0x3B    R0 0x03      DPL  0x31
0xFF  0xFF P2                            SP   0x47
0xFF  0xFF P1  PC 0x008F   PSW 0 0 0 0  1 0 0 0
0xFF  0xFF P0
                    Modify RAM
Data Memory         addr 0x00   0xC0 value

     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
00  C0 21 00 63 00 38 01 00 03 01 00 07 E2 32 02 00
10  3F 3F 00 78 00 30 00 31 00 32 78 A9 00 18 00 78
20  47 56 5F 6F 03 30 00 00 78 00 00 49 00 00 00 78
30  41 4F 31 6F 03 30 01 00 03 41 00 00 00 48 49 4A
40  F4 00 35 00 C0 C0 00 44 8C 00 00 40 31 00 08 01
50  3A 00 C0 C0 C0 C0 09 65 00 C0 C0 C0 C0 00 07 00
60  77 00 00 C0 C0 C0 10 00 00 A5 00 00 00 00 00 00
70  00 00 00 00 00 01 00 00 00 14 00 00 00 00 00 00
```

Copyright ©2005-2022 James Rogers

```
Executed 0x008C: MOV 0E0H,37H | Time: 423us
ØØ66|   MOV A,#5AH
ØØ68|   CJNE A,39H,Ø5H
ØØ6B|   MOV 39H,#41H
ØØ6E|   SJMP Ø5H
ØØ7Ø|   MOV A,39H
ØØ72|   INC A
ØØ73|   MOV 39H,A
ØØ75|   MOV C,Ø1H
ØØ77|   MOV ØAFH,C
ØØ79|   INC 36H
ØØ7B|   INC 37H
ØØ7D|   SJMP ØBEH
ØØ7F*   ORL 89H,#2ØH
ØØ82|   MOV 8DH,#ØFAH
ØØ85|   MOV 98H,#5ØH
ØØ88|   SETB 8EH
ØØ8A|   SETB 99H
ØØ8C|   MOV ØEØH,37H
ØØ8F|   JZ ØFBH
ØØ91|   DEC 37H
ØØ93|   MOV ØEØH,36H
ØØ96|   JZ ØFBH
ØØ98|   DEC 36H
ØØ9A|   SETB Ø2H
```

```
P0.7 1 Display-select Decoder CS|DAC WR
P0.6 1 Keypad Column 2
P0.5 1 Keypad Column 1
P0.4 1 Keypad Column 0
P0.3 1 Keypad Row 3
P0.2 1 Keypad Row 2
P0.1 1 Keypad Row 1
P0.0 1 Keypad Row 0
P1.7 1 LED 7|Seg. dp|DAC DB7|LCD DB7
P1.6 1 LED 6|Seg. g|DAC DB6|LCD DB6
P1.5 1 LED 5|Seg. f|DAC DB5|LCD DB5
P1.4 1 LED 4|Seg. e|DAC DB4|LCD DB4
P1.3 1 LED 3|... d|..DB3|..DB3|.. RS
P1.2 1 LED 2|... c|..DB2|..DB2|LCD E
P1.1 1 LED 1|Seg. b|DAC DB1|LCD DB1
P1.0 1 LED 0|Seg. a|DAC DB0|LCD DB0
P2.7 1 SW 7|ADC DB7
P2.6 1 SW 6|ADC DB6
P2.5 1 SW 5|ADC DB5
P2.4 1 SW 4|ADC DB4
P2.3 1 SW 3|ADC DB3
P2.2 1 SW 2|ADC DB2
P2.1 1 SW 1|ADC DB1
P2.0 1 SW 0|ADC DB0
P3.7 1 ADC RD|Comparator Output
P3.6 1 ADC WR
P3.5 1 Motor Sensor
P3.4 1 Display-select Input 1
P3.3 1 AND Gate Output|Display-se..t 0
P3.2 1 ADC INTR
P3.1 1 Motor Control Bit 1|Ext. UART Rx
P3.0 1 Motor Control Bit 0|Ext. UART Tx
```