

LESSON 6: WRITING PROGRAM1: CLOCK PROGRAM

Objectives:

1. Writing digital clock program.
2. Study internal timer.

The lesson will show how to write the program that displays time on terminal screen. The internal timer will be used to produce 10ms tick. The counting for second, minute and hour will then be a multiple of 10ms.

Let us formulate what is the idea of how to make digital clock using 8051SBC and terminal.

1. We will have 8051SBC sends time to terminal every second.
2. The display should show, `17:55:59`
3. Pressing key “m” will increment minute and key “h” will increment hour.
4. Onboard LED should blink every second.

EXERCISE 6-1:

1. Edit and assemble program below, run with command jump.

```
$mod51
#include(mypaulm2.equ)

led      equ P1.7
cr       equ 13

        dseg at 20h

flag1:   ds 1

        dseg at 50h

tick:    ds 1
sec100:  ds 1
sec:     ds 1
min:     ds 1
hour:    ds 1
timer3:  ds 1

        cseg at 8000h
        jmp start

        org 8100h
```

```

;---- initialization code -----
start:  orl tmod,#1 ; set timer0 to mode1
        setb tr0
        mov flag1,#0
        mov sec,#0
        mov min,#59h
        mov hour,#17h

;----- main loop running -----
main:   call wait_tick
        call update_clock
        call LED_on
        call LED_off
        call print_time
        jmp main

;----- subroutines -----
wait_tick: jnb tf0,$
           clr tf0
           orl th0,#0dch
           ret

; blink LED every second

blink_led: inc tick
           mov a,tick
           cjne a,#50,exit1
           mov tick,#0
           cpl led
exit1:    ret

update_clock:

           inc sec100
           mov a,sec100
           cjne a,#100,exit2
           mov sec100,#0

           setb flag1.0
           setb flag1.1

           mov a,sec
           add a,#1
           da a
           mov sec,a

```

```

    cjne a,#60h,exit2
    mov sec,#0

    mov a,min
    add a,#1
    da a
    mov min,a
    cjne a,#60h,exit2
    mov min,#0

    mov a,hour
    add a,#1
    da a
    mov hour,a
    cjne a,#24h,exit2
    mov hour,#0
exit2:
    ret

print_time:

    jnb flag1.0,exit_print_time
    clr flag1.0

print_time1:

    mov a,#cr
    call cout
    mov a,hour
    call phex
    mov a,#':'
    call cout
    mov a,min
    call phex
    mov a,#':'
    call cout
    mov a,sec
    call phex

exit_print_time:

    ret

LED_on: jnb flag1.1,exit_LED_on
        clr flag1.1
        setb flag1.2
        clr LED

```

```

exit_LED_on:
    ret

LED_off: jnb flag1.2,exit_LED_off
        inc timer3
        mov a,timer3
        cjne a,#5,exit_LED_off
        mov timer3,#0
        clr flag1.2
        setb LED

exit_LED_off:
    ret

    end

```

Now we will see the function for each portion of the program. First separate the whole program into four blocks.

Variable and constant declaration

Initialization

Main body

Subroutines

Variable and constant declaration

Let us look at variables and constants declaration

```

led equ P1.7
cr equ 13

    dseg at 20h

flag1: ds 1

    dseg at 50h

tick: ds 1
sec100: ds 1
sec: ds 1
min: ds 1

```

```
hour:    ds 1
timer3:  ds 1
```

LED represents port P1.7 and CR represents ASCII code 13 or 0DH.
Flag1 is declared at bit addressable location 20H. We can access the bit location by flag1.0, flag1.1, flag1.2, say. We use a given bit in flag for signaling among tasks.

The rest variables tick, sec100, sec, min, hour, and timer3 are declared as the byte variable located from address 50H.

Question

1. Write down the address for each variable.

Initialization

At start code, this portion will run only one time after we jump from monitor program. The code will set and load the initial value to the variable and hardware registers.

```
;---- initialization code -----
start:  orl tmod,#1 ; set timer0 to mode1
        setb tr0
        mov flag1,#0
        mov sec,#0
        mov min,#59h
        mov hour,#17h
```

Question

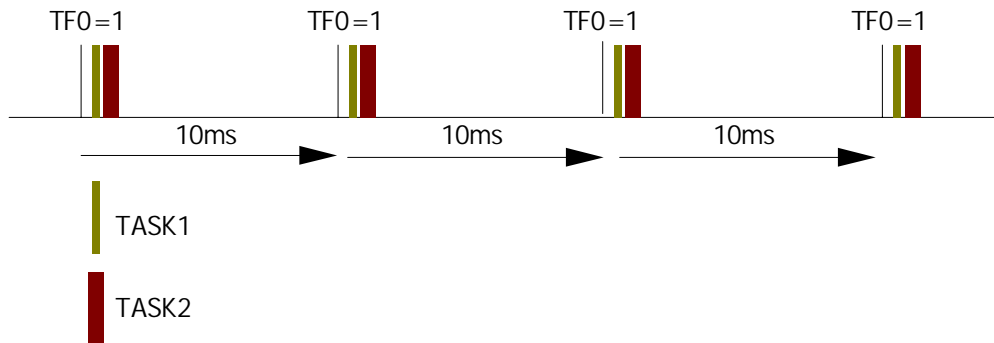
1. What are TMOD and TR0?

Main Body

The main body of the program is simple repeat running tasks in forever loop.

```
;----- main loop running -----
main:   call wait_tick
        call update_clock
        call LED_on
        call LED_off
        call print_time
        jmp main
```

The concept is the same as described in lesson 5. We use timer to force each task to execute every 10ms tick. However we can not have all tasks start running at the same time, since we have only one CPU. We then give a short period of time for each task execution. Recall the task running diagram again, we see now we have five tasks in the 10ms loop.



We can say that each task will repeat execution every 10ms. So the function that requires time control can use 10ms tick to count. We see that the 10ms tick is a critical time event, so we can not have task that runs forever loop. The solution to let task start execution is to use flag signaling. We will see how to use flag to force a given task to start execution on later.

Now let us study the details of the subroutines.

wait_tick

```
wait_tick: jnb tf0,$
           clr tf0
           orl th0,#0dch
           ret
```

Time0 is set to 16-bit counter. The reloaded count DC00H will set timer flag, TF0 every 10ms. When set, the code clears it and reloads DC00 to Timer0. This special task will exit when TF0 is set providing begin of execution the following tasks.

Update_clock

```
update_clock:

           inc sec100
           mov a,sec100
           cjne a,#100,exit2
           mov sec100,#0

           setb flag1.0
           setb flag1.1
```

```

    mov a,sec
    add a,#1
    da a
    mov sec,a
    cjne a,#60h,exit2
    mov sec,#0

    mov a,min
    add a,#1
    da a
    mov min,a
    cjne a,#60h,exit2
    mov min,#0

    mov a,hour
    add a,#1
    da a
    mov hour,a
    cjne a,#24h,exit2
    mov hour,#0
exit2:
    ret

```

This task updates current time by having sec100 to count 100 times for 100x10ms or one second.

When sec100 is equal 100, the sec variable will increment by adding to 1. The sec variable is BCD number. When sec variable is equal 60H, it was cleared to 0 and minute was incremented.

```

    mov a,sec
    add a,#1
    da a
    mov sec,a
    cjne a,#60h,exit2
    mov sec,#0

```

Flag1.0 and Flag1.1 are set every one-second. Flag1.0 signals print_time. Flag1.1 signals LED_on.

```

setb flag1.0
setb flag1.1

```

LED_on

```
LED_on: jnb flag1.1,exit_LED_on
        clr flag1.1
        setb flag1.2
        clr LED
exit_LED_on:
        ret
```

The function LED_on will execute only if Flag1.1 is set. If Flag1.1 is zero, it will exit.

If Flag1.1 sets, it will be cleared. Flag1.2 will be set for LED_off function, and the LED will turn on.

LED_off

```
LED_off: jnb flag1.2,exit_LED_off
        inc timer3
        mov a,timer3
        cjne a,#5,exit_LED_off
        mov timer3,#0
        clr flag1.2
        setb LED
exit_LED_off:
        ret
```

The function LED_off will run only if Flag1.2 is set. When Flag1.2 set, timer3 is incremented until 5 ticks, it will be cleared and onboard LED will turn off.

So with this method, the LED will flash every second with time on period equal 50ms!

print_time

```
print_time:
        jnb flag1.0,exit_print_time
        clr flag1.0

        mov a,#cr
        call cout
        mov a,hour
        call phex
```

```

        mov a,#':'
        call cout
        mov a,min
        call phex
        mov a,#':'
        call cout
        mov a,sec
        call phex

exit_print_time:

        ret

```

This function prints current time to terminal every one second. The printing was signaled by Flag1.0.

EXERCISE 6-2:

1. Modify the code to print DATE-MONTH-YEAR.

Now suppose we want to use PC keyboard to adjust time. Let add the function that reads character from serial port without forever loop inside. Remember if we use cin function, it will wait until SBUF has received ASCII character.

```

cin:  jnb ri,$ ; this line will wait until RI = 1
      clr ri
      mov a,sbuf
      ret

```

EXERCISE 6-3:

1. Add below subroutines to the main program.

```

get_command: jnb ri,exit_get_command
             clr ri
             mov a,sbuf
             mov command,a
             ret

exit_get_command:
             mov command,#-1
             ret

adjust_min:  mov a,command
             cjne a,#'m',exit_adjust_min
             mov a,min
             add a,#1

```

```

        da a
        mov min,a
        cjne a,#60h,adjust1
        mov min,#0
adjust1: call print_time1
        ret

exit_adjust_min:

        ret

adjust_hour: mov a,command
            cjne a,'#h',exit_adjust_hour
            mov a,hour
            add a,#1
            da a
            mov hour,a
            cjne a,#24h,adjust2
            mov hour,#0
adjust2: call print_time1
        ret

exit_adjust_hour:

        ret

```

2. Add the call instruction to these tasks also.

```

;----- main loop running -----
main:call wait_tick
      call update_clock
      call LED_on
      call LED_off
      call print_time
      call get_command
      call adjust_min
      call adjust_hour
      jmp main

```

3. Assemble and test run the program, PRESS key 'm' and 'h', see what happen on screen.
4. Modify the code to use key SPACEBAR to reset second to zero.