Lab 2: 8051-Based Timer and Stopwatch

ENGR 323: Microoprocessor Systems

Prof. Taikang Ning

Submitted by: Vishal Bharam and Bicky Shakya

Introduction

In the current digital age, microcontrollers have become an indispensible technological tool. Microcontrollers are single chip computers that are embedded into other systems, which can perform multiple complicated operations at the same time. In addition, microcontrollers have high integration of functionality, field programmability and are very easy to use. These features have helped microcontrollers to find a wide spectrum of applications, in the field of automobiles, video games, manufacturing industries and many others. Among all the microcontrollers in use today, the 8051 and its variations are considered the most popular.

Problem Statement

In this lab, we will use the popular 8-bit 8051 microcontroller to design a system to perform time-count. The timer is equipped with four 7-segment displays that will count from 00:00 to 59:59 and then reset back to counting from 00:00.

The extension of the first lab will require us to include a stop-watch mode in our system. The final system should meet the following requirements:

- To count minutes and seconds from 00:00 (out of reset) to 59:59 and repeat the counting after every full hour (completed in the first lab)
- To allow external controls/interrupts to emulate a stopwatch which can reset, start and freeze counts by the 100th of a second.

Specific Design Goals

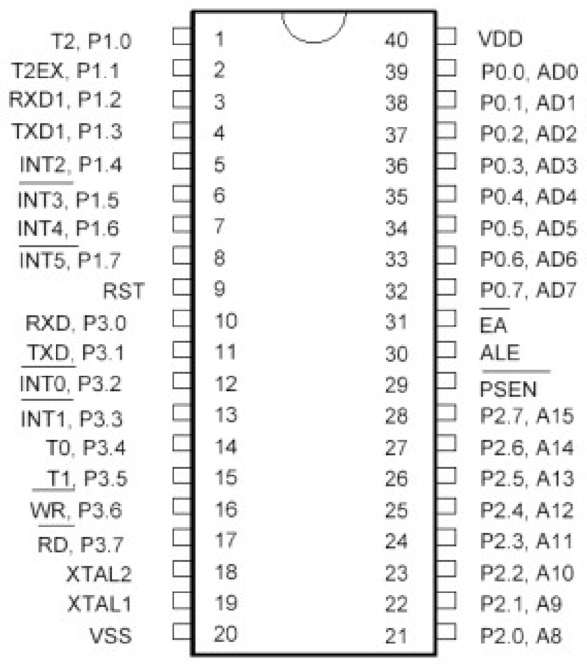
As mentioned already, the primary objective of the first laboratory is to design a digital clock on a 4 LED display using 8-bit 8051 microcontroller. The target system is expected to perform sixty seconds counts, i.e., for example counting from 00:00 to 00:59 in one second interval and repeating the counts one second after it reaches 00:59. For displaying the time, four LED based displays will be used, among which two digits are used for displaying minutes and remaining two for seconds.

This lab is divided into two different sections to make the whole process easier. The objective of the first section is to complete the circuit consisting of four seven-segment decoder, 3-8 decoder, hex inverter and the 4 LED display and check it using the inputs from CADET board. After displaying the desired outputs on the LED using a CADET board, the main circuitry needs to be completed with the 8051 microcontroller, a latch and EEPROM. Using assembly language, the EEPROM is to be programmed to display the digital clock counting from 00:00 second to 59:59 minutes and recount again after resetting back to 00:00. The four 7-SEG LED's require a higher power and many I/O pins. TO overcome this drawback, a time-division-multiplexing (TDM) dynamic control method is used instead. In this approach, output pins are shared by 7-SEG displays and each segment will be refreshed frequently enough to make it seemly as statically lit to human eyes.

For the second lab, we need to incorporate a stop watch mode in the same circuitry that was used to display the hour-timer. The stop-watch display will need to have the last two digits for milliseconds (count ranging from 00 to 99). After 99 milliseconds have passed, the next digit needs to increment for seconds. The seconds will need to range from 00 to 59. As we are using the same circuitry and trying to keep hardware modifications to the minimum, we will be using the two external interrupts on the 8051 to switch between the timer and the stop watch mode, start the stopwatch and freeze the timer while it is running. The same concepts of time-division multiplexing will be used for the display. Modifications will only be made to the coding section of the system, especially in the way the external interrupts are used to execute specific subroutines. Firstly, the TMOD, TCON, IE and IP SFR's should be properly configured and interrupts TF0, TF1, IE0 and IE1 should be activated. Then, the first activation of an external interrupt should invoke an external interrupt routine to change the updating timer overflow interrupt service routine to an interval of a hundredth of a second. Also, it is to be noted that when the stop-watch mode is activated, the timer should be simultaneously running/counting (in the background)

Hardware:

The 8051 microcontroller, also known as the MCS-51, will be used in this project. Given below is the layout/pin assignment of the 8051 microcontroller.



The 8051has four ports, each port being equipped with 8 pins. Port 0 can be used as input or output pins (depending on how we configure hardware i.e. connect pull up resistors). But mainly, pins on Port 0 can be programmed to use both address and data (this is decided by the ALE pin; ALE = 0 indicates Port 0 be used for data so, it would be D0-D7. ALE = 1 would set the pins for both address and data, provided it is put through a latch (the latch, the **74HC573**, is used in our system to demultiplex the address and data signals on P0.0 – P0.7).

Port 2 will also be used in this design. Port 2 can be configured as either input or output pins by programming. Here, we will only use P2.0 – P2.2. These pins will be connected directly to the EEPROM without putting them through a latch.

The EEPROM we are using is 2 kilobytes ($2 * 2^{10} = 2^{11}$). So, we will require 11 address bits and 8 data bits (16 Kilobits = 2 Kilobytes * 8). For the 11 address bits, we have 8 pins coming through Port 0. The remaining 3pins will be used from Port 2, which justifies selecting P2.0 – P2.2.

Below are the descriptions of some of the other pins that will be used in this project.

- PSEN and ALE: The pin PSEN from the 8051 is connected to the output enable (OE) pin on the EEPROM. When PSEN is low, the EEPROM is enabled (so PSEN is active low).

  In the system, the 8051 initially sends out address bits from P0.0 – P0.7 to the latch. All of the events are clocked and during a clock cycle, when ALE goes high, the latch takes these address bits and passes them on to the EEPROM to fetch the data stored at the sent address locations from the 8051. In the following clock cycle, when the ALE goes to low and at the same time, PSEN goes to low, the latch switches to collecting data bits from the EEPROM and passing it on to the 8051.

- EA: This pin is set to low to select the external memory (EEPROM).

  XTAL1, XTAL2: These are the two pins that are used to connect an external clock to the 8051 microcontroller. The 8051 does have an on-chip oscillator but it requires the external clock to run it. We connected a 12 Hz crystal to the 8051. This sets a clock speed of the 8051 to 1 Hz (divide crystal frequency by 12 as the 8051 takes 12 clock pulses to execute one single instruction or one machine cycle).

- RESET

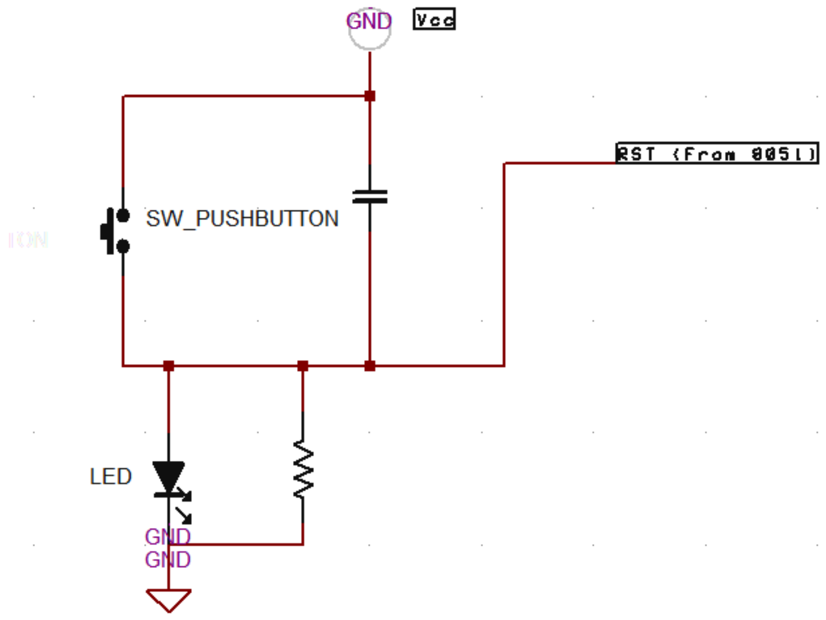


Fig. Reset Circuitry

The reset circuitry is shown in the picture above. The RST pin on the 8051 (Pin 9) is connected to a pushbutton and a capacitor (100 uF) in parallel. When the pushbutton is depressed, the RST from the 8051 is connected to Vcc, turning it to high. This enables a reset of the system. When the pushbutton is not depressed, the RST pin is connected to ground, which means that the system needs to stay in its current state and should not reset. An LED was also connected in parallel with the pull-down resistor so that it would light up whenever the pushbutton was pressed.

All the other pins are left unconnected as they are not needed for our counter system.

Other Chips for Use in the System:

- DM74LS138 (3 to 8 Multiplexer): This chip was used to serve as a selection line for the four seven-segment display. The system would actually need a 2 to 4 mutliplexer but we used a 3 to 8 in its place, while tying the third input to ground. The two inputs would be connected to P0.4 and P0.5 from the 8051. The output from those pins would determine which of the four 7 segment displays we would be using at a given instant. For example, when both P0.4 and P0.5 would be low, the first 7 segment display would be used where as both pins being high would mean the fourth display would be used. Additionally, the G1 pin was tied to high and the G2 pins were tied to ground as per the truth table for the multiplexer.

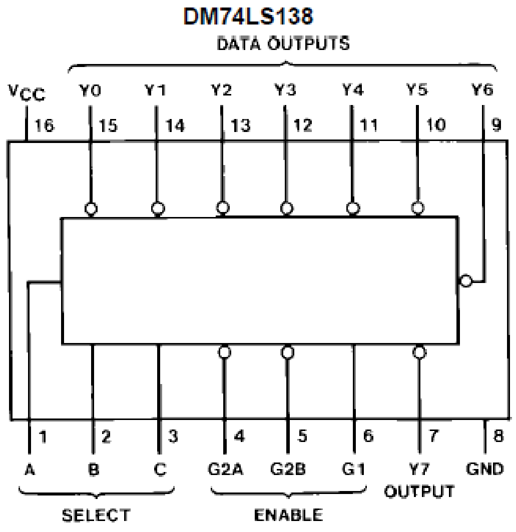


Fig. Pin Configuration for the DM74LS138

- DM7447A (BCD To 7-Segment Display): This decoder chip would be used to display numbers on the individual seven-segment displays. The four inputs (the 4 digit BCD or A – D in the pin configuration diagram below) were connected to P0.0 to P0.3 on the 8051 and the outputs were connected to the display through a pull-up resistor. Also, the Lamp test and BI/RB0 pins were tied to high as per the data sheet of the chip.

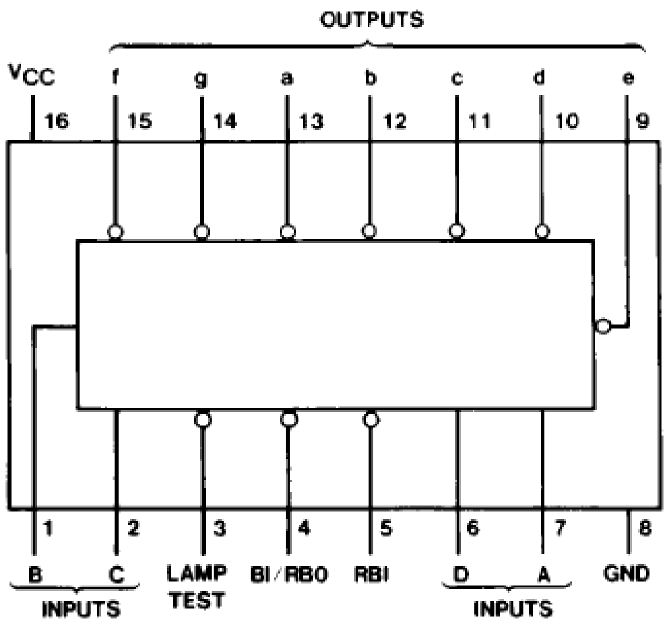


Fig. Pin Configuration for the DM7447A Decoder

- LDQ-M514RI (Four Digit Seven-Segment Display):

This display is used in showing the four digits of our timer.

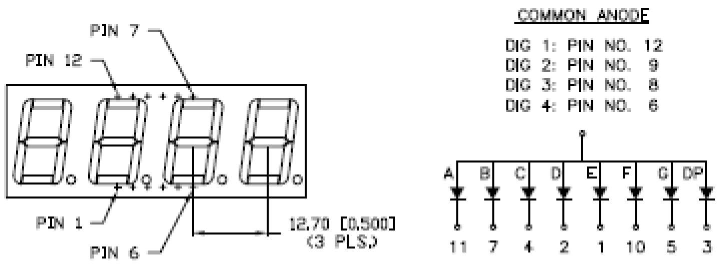


Fig. Pin Configuration for the 4 Digit 7-Segment Display

The pin configuration suggests that pins 6,8,9 and 12 are used for selecting among the four digits. These pins are connected to the output of the 3 to 8 demultiplexer. The other pins connect to the outputs of the BCD to 7-Segment Display decoder. Depending on the BCD output from the decoder, the necessary sections of the seven segment display will be lit up to display the numbers on our timer.

- Vdd : This pin is connected to +5V and there is another pin, called Vss that is connected to ground. These pins provide power to the 8051 (which usually draws about 500 mA).
- EEPROM (MC28C16A): The external memory we used in this system was the MC28C16A EEPROM, which has a capacity of 2 Kilobytes.

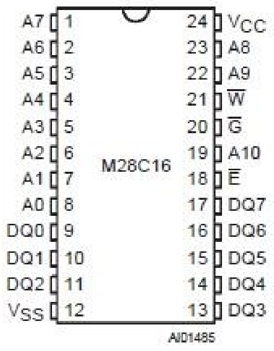


Fig. Pin Configuration for MC28C16A

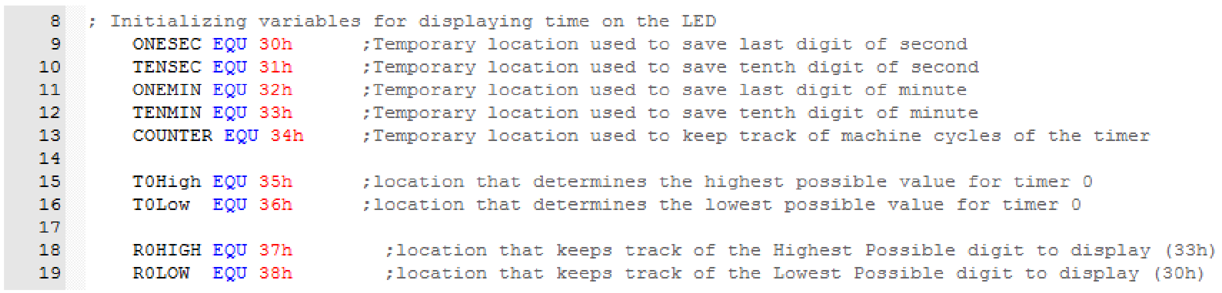
Pins A0 – A10 are connected to Ports 0 and 2 from the 8051 (through latch for pins from Port 0). These connections are important for relaying the data/address to and from the 8051. The Output Enable (G) pin is connected to the PSEN pin from the 8051. This pin basically determines whether we are using the EEPROM or not (PSEN = 0 indicates EEPROM enable). The pins D0 – D7 are connected back to P0.0 – P0.7 on the 8051 according to the system schematic provided. Also, the chip enable (E) is connected to GND so that the EEPROM is enabled at all times (active low). Vcc is connected to +5V and Vss is connected to GND to power up the EEPROM.

- **Pushbuttons on CADET Board**
  For the second lab, one pushbutton on the CADET board will be connected to pin 3.2 (external interrupt 0) and the other pushbutton will be connected to pin 3.3 (external interrupt 1) on the 8051. These pushbuttons are naturally shorted to ground but upon pushing the button, a HIGH signal (+5V) will be sent to the respective pins to which they are connected (P3.2 andP3.3). These two buttons will be used to invoke external interrupts.

Software:

This lab assignment was the extension of the previous lab, which was 8051-based digital clock. The system was to be expanded by adding two external interrupts to allow flexibility to emulate stopwatch function. Though, the primary objective of this lab was to emulate stopwatch function, we also used the previously built digital clock. Thus, the syntax for the digital watch is just expanded using the external interrupts. The brief flowchart of the syntax is given below:

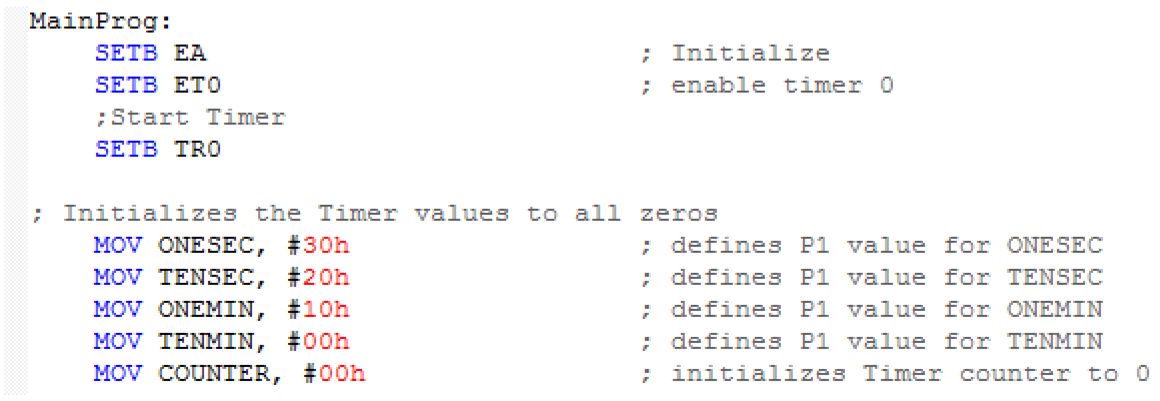
In order to make the programming aspect easier, we used the ‘EQU’ notation to assign various variables to different memory addresses. As both of us had a bit experience with languages using variables, variable assigning made our job a little clear. Using indirect addressing, a helpful feature in assembly, we were able to assign variables as given below.



```
; Initializing variables for displaying time on the LED
    ONESEC EQU 30h      ;Temporary location used to save last digit of second
    TENSEC EQU 31h      ;Temporary location used to save tenth digit of second
    ONEMIN EQU 32h      ;Temporary location used to save last digit of minute
    TENMIN EQU 33h      ;Temporary location used to save tenth digit of minute
    COUNTER EQU 34h     ;Temporary location used to keep track of machine cycles of the timer

    T0High EQU 35h      ;location that determines the highest possible value for timer 0
    T0Low  EQU 36h      ;location that determines the lowest possible value for timer 0

    R0HIGH EQU 37h        ;location that keeps track of the Highest Possible digit to display (33h)
    R0LOW  EQU 38h        ;location that keeps track of the Lowest Possible digit to display (30h)
```

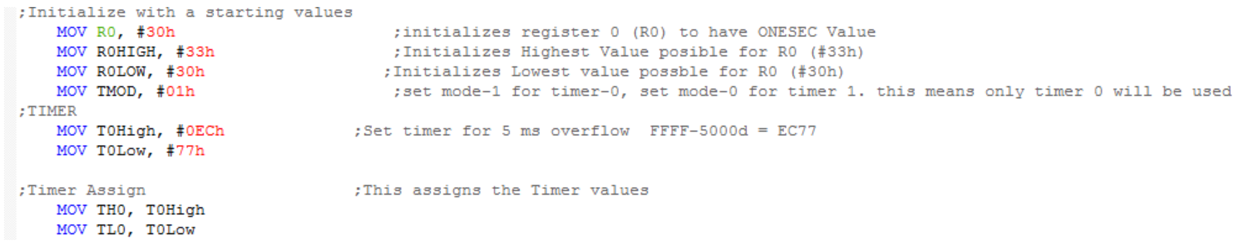
As we can see in the subroutine above, temporary memory addresses are being used to save each digit on the 4 7-SEG LED display. We also initialized all the bit addressable bits in the special function registers, TMOD and TCON, along with counter, which we will be needed in our program. Memory addresses from 30h to 33h are used sequentially to initialize from last digit of second to highest digit of minute. Memory address 35h and 36h are initialized to the highest and smallest value of timer 0 respectively, while address bytes 37h and 38h are initialized to the highest and highest value of register 0 (R0), which is used to shift the digit so that the display can be refreshed frequently enough on the 4 7-SEG LED display.

After assigning the temporary memory addresses, the timer is enabled and is ready to count. As given in the figure below, values for the all digit on the 4 7-SEG display are initialized to some hex number which is to represent zero as we want the clock to start from 00:00. These initialized values are used later in the ISRTF0 subroutine. In addition, the counter is also set to zero as well.



```
MainProg:
    SETB EA                                 ; Initialize
    SETB ET0                                ; enable timer 0
    ;Start Timer
    SETB TR0

; Initializes the Timer values to all zeros
    MOV ONESEC, #30h                        ; defines P1 value for ONESEC
    MOV TENSEC, #20h                        ; defines P1 value for TENSEC
    MOV ONEMIN, #10h                        ; defines P1 value for ONEMIN
    MOV TENMIN, #00h                        ; defines P1 value for TENMIN
    MOV COUNTER, #00h                       ; initializes Timer counter to 0
```

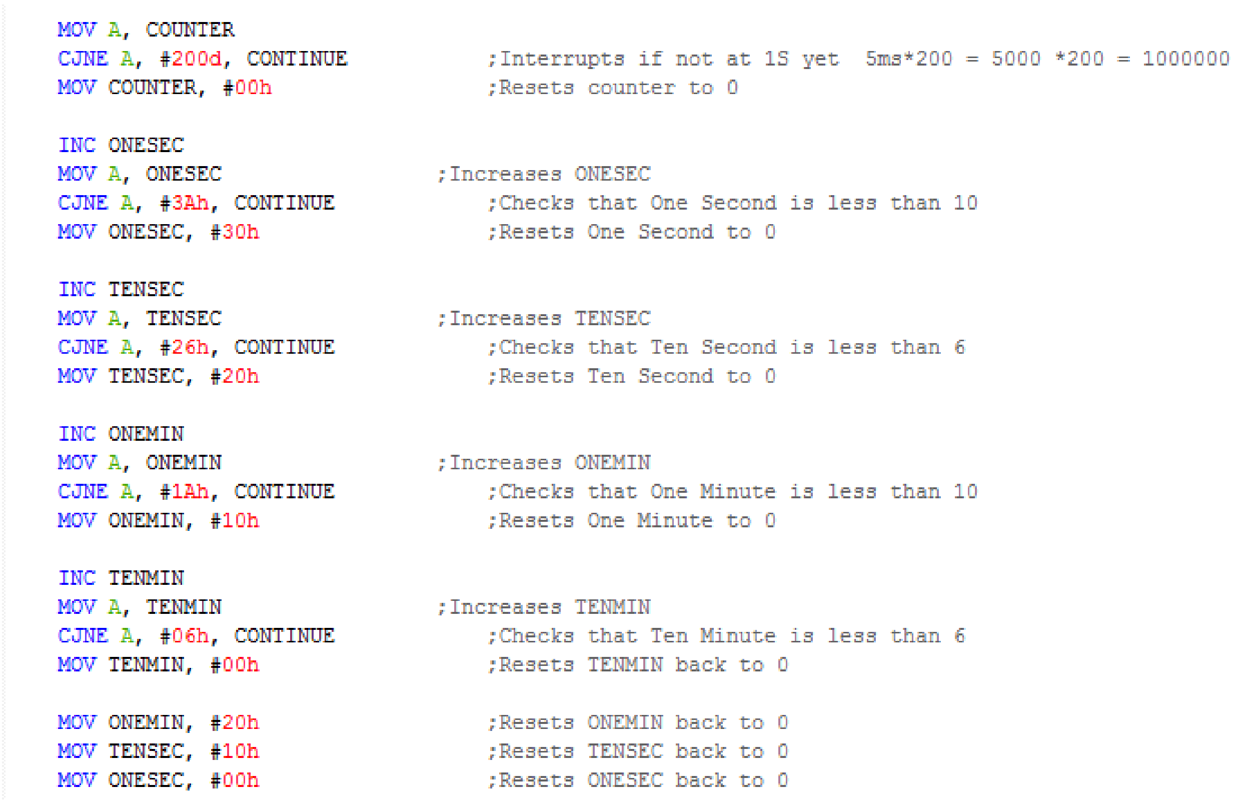
Register 0 (R0) is initialized to the value of last digit on LED display consisting on 0-9 seconds and first digit, which holds the value for 0-5 minutes. In order for the display to be refreshed, a 5ms delay is being placed between each refresh so that the display will not be flickering and it will seem statically lit to the eyes. Thus, value of **EC77h (FFFFh-5000d)** is then loaded to the timer 0 as shown below. Furthermore, the TMOD is also set to 01, which means timer0 is enabled while timer1 is disabled as it is not being used.



```
;Initialize with a starting values
    MOV R0, #30h                     ;initializes register 0 (R0) to have ONESEC Value
    MOV R0HIGH, #33h                 ;Initializes Highest Value posible for R0 (#33h)
    MOV R0LOW, #30h                 ;Initializes Lowest value possble for R0 (#30h)
    MOV TMOD, #01h                   ;set mode-1 for timer-0, set mode-0 for timer 1. this means only timer 0 will be used
;TIMER
    MOV T0High, #0ECh           ;Set timer for 5 ms overflow  FFFF-5000d = EC77
    MOV T0Low, #77h

;Timer Assign                   ;This assigns the Timer values
    MOV TH0, T0High
    MOV TL0, T0Low
```

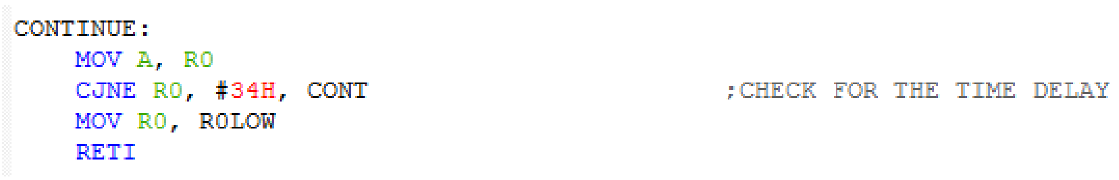
Then, the timer 0 overflow interrupt subroutine is set. As we know, the timer0 overflow subroutine (Timer 0) is set to overflow every 5ms. When the timer overflows, the Timer Overflow Interrupt0 (ISRTF0) flag is set and the 000Bh memory jump address is called, which immediately jumps to the ISRTF0 subroutine and executes it. In this subroutine, we increase the value in R0, which controls which digit is being displayed. Because the subroutine is called every 5ms, our program cycles through the digits every 5ms, giving the illusion that all four digits are being displayed at the same time. It is also necessary to reset the timer flow so that it will overflow after the next 5ms.

The values that were used to initialize each digit of the 4 7-SEG LED display are used to create loops that will return the incremented values from 0 to 9 the lowest digits (2nd and 4th) of seconds and minutes values and 0-5 for the highest digits (1st and 3rd) of second and minutes values. Furthermore, we know that delay was set for 5ms, the counter variable was repeated for 200 times (200*5000 = 1000000µsec = 1sec) to shift the values from digit to digit. After reaching the value of 59:59 minutes, the display is reset back to 00:00 second as shown below.



```
    MOV A, COUNTER
    CJNE A, #200d, CONTINUE          ;Interrupts if not at 1S yet  5ms*200 = 5000 *200 = 1000000
    MOV COUNTER, #00h                ;Resets counter to 0

    INC ONESEC
    MOV A, ONESEC                ;Increases ONESEC
    CJNE A, #3Ah, CONTINUE           ;Checks that One Second is less than 10
    MOV ONESEC, #30h                 ;Resets One Second to 0

    INC TENSEC
    MOV A, TENSEC                ;Increases TENSEC
    CJNE A, #26h, CONTINUE           ;Checks that Ten Second is less than 6
    MOV TENSEC, #20h                 ;Resets Ten Second to 0

    INC ONEMIN
    MOV A, ONEMIN                ;Increases ONEMIN
    CJNE A, #1Ah, CONTINUE           ;Checks that One Minute is less than 10
    MOV ONEMIN, #10h                 ;Resets One Minute to 0

    INC TENMIN
    MOV A, TENMIN                ;Increases TENMIN
    CJNE A, #06h, CONTINUE           ;Checks that Ten Minute is less than 6
    MOV TENMIN, #00h                 ;Resets TENMIN back to 0

    MOV ONEMIN, #20h                 ;Resets ONEMIN back to 0
    MOV TENSEC, #10h                 ;Resets TENSEC back to 0
    MOV ONESEC, #00h                 ;Resets ONESEC back to 0
```

As seen in the syntax above, 'CONTINUE' function is used to display the digit on the 4 digit 7-SEG display. The position of the digit determined by the register R0, so R0 is moved to accumulator (A). The position of digit is compared and incremented using 'CJNE' and the values are retuned using 'RETI' as shown below:



```
CONTINUE:
    MOV A, R0
    CJNE R0, #34H, CONT                        ;CHECK FOR THE TIME DELAY
    MOV R0, R0LOW
    RETI
```

Implementation:

After sketching the schematic for the 8051 system, the whole 8051 system was wired on the PC board using the IC sockets. The whole system included 8051 connected to the latch and EEPROM along with an external circuit with BCD 7-segment decoder (7447), 3-8 decoder (138) and four 7-SEG display (LDQ-M514RI). All of the chips were powered in parallel and they were given common ground. The program written in Keil µVision was loaded into the EEPROM through EMP21 Device Programmer.

For the second lab, the only hardware modification we made was that we connected Pins 3.2 and 3.3 on the 8051 to the two pushbuttons on the CADET board. This was done because on pressing the pushbutton, we would invoke the external interrupts for activating the stop-watch (switching modes) and also freezing the stop-watch count. Normal wire wrapping to make the connections to the external interrupt pins.

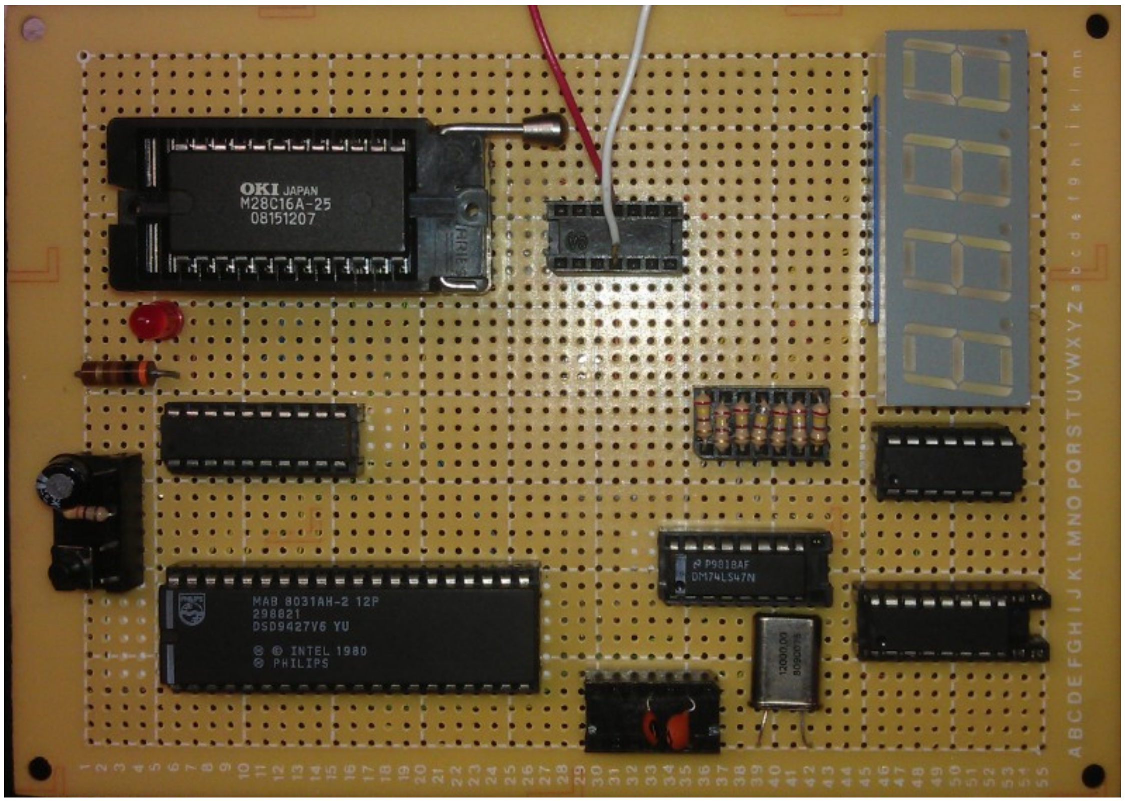


Fig. View of the Overall 8051 System on a PC Board

During the process of designing the system, we encountered our first main roadblock when we tried to test the part of our circuit for the 4 digit seven-segment display. We did not use the 8051 at this stage. We connected the pins (corresponding to P0.0 – P0.5 in the 8051) from the 8051 socket into six separate logic switches on the CADET board (4 for the BCD and 2 for selecting the digit to be displayed). The output from those logic switches were connected to our circuit and those logic switches were varied to display numbers on the four-digit seven-segment display. Unfortunately, the only output we were getting on the display was a zero on the first digit. We later realized that we had wired the 4-digit seven segment displays in the opposite way, leading to the error. After fixing that problem, we finished wiring the entire circuit on the PCB board. Later on, we encountered another problem. After programming the EEPROM and loading it on the system, we found that no matter what code we uploaded, the display was either only a zero on the first digit or no display at all. At times, we found that several of the wire wraps we had made had either shorted or came off loose. But nonetheless, our system still did not respond to the code. We finally figured out (with help from Prof. Ning of course) that we had wired up our reset circuit incorrectly. When the push button was depressed, the Reset pin on the 8051 should have been set to high (as it would be shorted to Vcc. But it was not so; instead, Reset pin showed a voltage close to zero when the pushbutton was depressed. We then rewired our reset circuit properly after which our system responded to the program stored on the EEPROM.

For the second lab, the major problem we encountered was the problem of refreshing the display for the stopwatch mode. We noticed that even after putting in the correct delay values, our stopwatch mode would flicker too much and make it hard for us to read the count. We found out in the end that re-initializing the values for delay in the interrupt service routine (besides initializing at the start of the program) fixed the problem.

Also, we initially faced a problem with the stopwatch count. Upon switching to the stopwatch mode, we observed that the stopwatch count would begin from the count that the timer had left off, instead of starting from 00:00. After much time, we figured out the simple mistake we had made; we had not re-initialized the values for the display digits in the interrupt service routine. After doing that, all the display digits were reset before starting the stopwatch count, as per the design requirement.

It also took a while to figure out how to freeze the stopwatch count. In the end, we ended up using the bit-addressable registers we had access to, in the 8051. We used a bit from the 20h register in the bit memory space of the 8051 Internal RAM. This bit was initially cleared at the start of the program but then, in our interrupt service routing for external interrupt 1, we defined the statement 'CPL BIT1'. This meant that whenever the pressing of the pushbutton would call on the external interrupt 1, BIT1 would be set. In our program, we had a JNB statement where, if BIT1 was not set, the program would go towards counting the stopwatch. But if BIT1 was set (through external interrupt 1), the program would skip the stopwatch counting part, which would effectively 'freeze' the value of the stopwatch at what it was before calling on external interrupt 1. We initially tried another approach too, where the stopwatch subroutine was called in ISRTF0 using the 'AJMP' command. We were thinking about disabling the ISRTF0 through

ISRIE1 to freeze the stopwatch count. This however, did not work well at all. But the second approach we used (with BIT1, JNB) worked quite well in the end and allowed us to freeze the stopwatch by pressing the External Interrupt 1 pushbutton, while the stopwatch was running.

Results and Discussion

This lab gave us a great understanding of the 8051 microcontroller and a sample timing system that could be built around it. One of the most challenging parts of the whole project was to come up with a specific technique to write the assembly code. After putting our ideas and techniques into a flowchart, the coding process became much easier. As discussed earlier, we encountered some problems in the actual circuit which prevented us from getting the required output. However, after spending a lot of time on debugging, we got everything working perfectly. Our system was successfully able to count from 00:00 to 59:59 after which it reset to 00:00 and started counting again. Thus, our system met its design goals.

The second lab was also successfully completed. We successfully expanded our preexisting code for the timer to include the stopwatch mode, by coding for the external interrupts. Different interrupt service routines were created for the stop watch mode and were successfully invoked by activating the pushbuttons. In the end, our expanded system was able to display the timer mode properly, counting from 00:00 to 59:59 (MM:SS) and was also able to switch to stopwatch mode, where the count started from 00:00 (SS:MsMs) and kept on going until 'freeze' was evoked by another interrupt service routine (controlled by the pushbutton).

So, to conclude, we successfully built an 8051-based system that incorporated several elements such as time-division multiplexing, external interrupts and timer overflow interrupts to operate a timer as well as a stopwatch.

References

- Datasheets for respective chips
- http://www.8052.com/tuttimer.phtml