# ISTANBUL TECHNICAL UNIVERSITY

# COMPUTER ENGINEERING DEPARTMENT

## BLG 351E

## MICROCOMPUTER LABORATORY EXPERIMENT REPORT

**EXPERIMENT NO** : 7-8

**EXPERIMENT DATE** : 04.12.2019

**LAB SESSION** : WEDNESDAY - 13.30

**GROUP NO** : G10

## GROUP MEMBERS:


150170062 : Mehmet Fatih YILDIRIM

150180704 : Cihat AKKİRAZ

150180705 : Batuhan Faik DERİNBAY

150180707 : Fatih ALTINPINAR


**FALL 2019-2020**

# Contents

# 1 INTRODUCTION

In this experiment, 16x2 Dot-matrix LCD display and a random generator is implemented. While doing this implementation, experience and knowledge from old experiments are used. Also given informations and codes are used to understand how manipulating LCD display via using MSP430G2553 microcontroller.

# 2 MATERIALS AND METHODS

This experiment is conducted via using MSP430G2553 microprocessor. This microprocessor is programmed using Code Composer Studio according to desired tasks on the experiment handout. During coding below sources are used:

- MSP430 Education Board Manual [1]
- MSP430 Architecture Chapter 4 [2]
- MSP430 Instruction Set [3]
- Supplementary Chapter 6 General Purpose [**?**]
- MSP430 User Guide - Chapter 8 [**?**]

## 2.1 Part 1

In the first part of the experiment, the team was asked to implement a print function which uses char arrays (See Figure 2.1) as inputs and show them on the LCD display. Also while displaying this array, it is asked that \n and \0 characters should be interpreted and create outputs according to these interpretations. (See Figure 1).


Figure 1: Display on the 16x2 lcd asked string

```
1           .data
2 string    .byte "ITU␣-␣Comp.␣Eng.",0Dh,"MC␣Lab.␣2019",00h
3
```

Figure 2: String asked to display on the LCD

To display and manipulate values on the LCD display, some configurations should be made. In these experiments, aforementioned configurations are already given to the team and the code for only to send data and display string on the LCD was asked. Given subroutines are listed below and the codes of these subroutines are not included in the report. In this report, only the parts of the code that are written by the team are included.

1. initLCD
2. sendCMD
3. sendDATA (This part was edited by the team)
4. delay
5. trigerEN

Let us go through the code and analyze the implementation.

- Line 1: Address of string is moved to R10.
- Line 2-3: Controlling whether end of the line for current char. If the current element is '0dh' jump to execute newline subroutine.
- Line 4-5: Controlling whether end of the sequence for current char. If the current element is '00h' jump to execute endseq subroutine.
- Line 6-7: Moving char value on the R10 register to R5 for SendData subroutine and calling these subroutine.
- Line 8-9: Delay for sending data to LCD.
- Line 10-11: Incrementing R10 to process next char in the string array and jumping to beginning of the loop to process it.
- Line 13: Point to next character in the array.
- Line 14-15: Load the command for newline in R5 and send it.

- Line 16-18: Wait for the execution of newline operation, then return back to main loop.
- Line 20: Ends the sequence by jumping to infinite loop stop.
- Line 22: Puts the device in write mode so device is ready for receiving data.
- Line 23-30: Sends the upper nibble first then shifts the data 4 times and sends the second nibble.
- Line 31-32: Puts the device in read mode since no more data will be sent in the given function. Then returns from subroutine.
- Line 34-36: Infinite stop loop. Calls itself every 100 microseconds.

```
1                   mov.w #string,  R10
2   loop            cmp.b   #0dh,   0(R10)
3                   jz      newline
4                   cmp.b   #00h,   0(R10)
5                   jz      endseq
6                   mov.b   @R10, R5
7                   call    #SendData
8                   mov     &Delay100us, R15
9                   call    #Delay
10                  add.w   #01b,   R10
11                  jmp loop
12
13  newline         add.w   #01b,   R10
14                  mov.b   #011000000b, R5
15                  call    #SendCMD
16                  mov     &Delay100us, R15
17                  call    #Delay
18                  jmp             loop
19
20  endseq          jmp stop
21
22  SendData        bis.b #080h,  &P2OUT
23                  mov.b R5, &P1OUT
24                  call #TrigEn
25                  rla R5
26                  rla R5
27                  rla R5
28                  rla R5
29                  mov.b R5, &P1OUT
30                  call #TrigEn
31                  bic.b   #080h,  &P2OUT
32                  ret
33
34  stop            mov.b   &Delay100us,    R15
35                  call #Delay
36                  jmp stop
```

Figure 3: Sending Data to the LCD Display - Part 1

## 2.2 Part 2

In this part, a random generator is implemented and generated values are displayed on the 16x2 LCD dot display.

To accomplish this task the parts added or modified can be explained as following.

- Line 6: Rather ending the sequence when $00h$ character received, program jumps to subroutine called *writenumber* which will print the number on the LCD display.
- Line 89-120: In the interrupt service routine, following calculations are made to obtain a random number on $R9$. $R11$ holds the seed value, $R13$ is $x$ and $R10 = w$.
- Line 91-96: Taking square of the $x$ is done by calling multiplication subroutine by giving parameters as $x$ and $x$ again. Which returns $x^2$.

$$x = square(x)$$

- Line 98-100: $x = x + (w = w + s)$ is implemented by these lines. It sums the corresponding registers in order to obtain $x$ value.
- Line 101-110: $r = (x >> 4)|(x << 4)$ is calculated by using a temporary register $R6$ which was pushed to stack at the beginning of the interrupt service routine.
- Line 113: Increasing seed which is not an ideal thing to be done in order to obtain random numbers.
- Line 115: Sets $R12$. This register is checked in the stop sequence. If a new number comes system start to write to the LCD all over again.
- Line 117-120: Popping temporary register to its previous value and returning from interrupt.
- Line 22-56: In order to print the random number in $R9$, every digit has to be converted to character at first. This is done as following:
  - Line 23-28: Finding hundreds digit by dividing $R9$ by 100. Result is going to be hold on $R7$.
  - Line 30-43: Tens digit is found by the formula given below. This value is then loeaded to $R7$:

$$TensDigit = RandomNumber/10 - HundredsDigit \times 10$$

  - Line 44-54: Units digit is found by the same way, then it is stored inside $R8$.

$$UnitsDigit = RandomNumber - HunderdsDigit \times 100 - TensDigit \times 10$$

- Line 54-56: The numbers of every digit is found but these numbers have to be converted to char from int. Since LCD display uses ASCII, 48 which is the value of ASCII 0 is added to every register. This ensures all registers will have corresponding char values to be printed.
- Line 57-68: Digit by digit all numbers printed on the screen

- Line 72-82: Enables $RS$ which allows data transmission. $R5$ is loaded with the data then LCD Display is triggered. Which takes upper nibble then lower. $R5$ shifted towards left in order to send lower nibble to LCD display, which only reads from $7^{th} - 4^{th}$ bits of $P1OUT$.

```
;r10= address pointer
                mov.w   #string,        R10
loop            cmp.b   #0dh,   0(R10)
                jz      newline
                cmp.b   #00h,   0(R10)
                jz      writenumber
                mov.b   @R10, R5
                call    #SendData
                mov     &Delay100us, R15
                call    #Delay
                add.w   #01b,   R10
                jmp     loop

newline         add.w   #01b,   R10
                mov.b   #011000000b, R5
                call    #SendCMD
                mov &Delay100us, R15
                call    #Delay
                jmp     loop

; R9 = incoming generated number R10 = temporary
writenumber     ; Find hundreds digit
                sub.w   #02h,   sp
                push    #100d
                push    r9
                call    #Div_func
                add.w   #04h,   sp
                pop     r6

```

Figure 4: Code 1/4 - Part 2

```
30                  ; Find tens digit
31                  sub.w   #02h,   sp
32                  push    #10d
33                  push    r9
34                  call    #Div_func
35                  add.w   #04h,   sp
36                  pop     r7
37                  sub.w   #02h,   sp
38                  push    #10d
39                  push    r6
40                  call    #Mul_func
41                  add.w   #04h,   sp
42                  pop     r10
43                  sub.b   r10,    r7
44                  ; Find units digit
45                  add.b   r7,             r10
46                  sub.w   #02h,   sp
47                  push    #10d
48                  push    r10
49                  call    #Mul_func
50                  add.w   #04h,   sp
51                  pop     r10
52                  sub.b   r10,    r9
53                  mov.b   r9,             r8
54                  add.b   #048d, R6
55                  add.b   #048d, R7
56                  add.b   #048d, R8
57                  mov.b   R6, R5
58                  call    #SendData
59                  mov     &Delay100us, R15
60                  call    #Delay
61                  mov.b   R7, R5
62                  call    #SendData
63                  mov     &Delay100us, R15
64                  call    #Delay
65                  mov.b   R8, R5
66                  call    #SendData
67                  mov     &Delay100us, R15
68                  call    #Delay
69
```



Figure 5: Code 2/4 - Part 2

```
70                  mov.b   #000h,  &P2DIR
71                  jmp             stop
72  SendData        bis.b #080h,  &P2OUT
73                  mov.b R5, &P1OUT
74                  call #TrigEn
75                  rla R5
76                  rla R5
77                  rla R5
78                  rla R5
79                  mov.b R5, &P1OUT
80                  call    #TrigEn
81                  bic.b   #080h,  &P2OUT
82                  ret
83  stop            mov.b   &Delay100us,    R15
84                  call    #Delay
85                  cmp.b   #01h,   r12
86                  jz              InitLCD
87                  jmp     stop
88
```

Figure 6: Code 3/4 - Part 2

```
89  ISR             dint
90                    push    r6
91                    sub.w   #02h,   sp
92                    push    r13
93                    push    r13
94                    call    #Mul_func
95                    add.w   #04h,   sp
96                    pop     r13
97
98                    add.b   r11,    r14
99                    add.b   r14,    r13
100                   mov.b   r13,    r9
101                   rra.b   r9
102                   rra.b   r9
103                   rra.b   r9
104                   rra.b   r9
105                   mov.b   r13,    r6
106                   rla.b   r6
107                   rla.b   r6
108                   rla.b   r6
109                   rla.b   r6
110                   bis.b   r6,     r9
111
112                   ; Change seed
113                   add.b   #01h,   r11
114                   ; I was in an interrupt flag
115                   mov.b   #01h,   r12
116
117                   pop     r6
118                   clr     &P2IFG
119                   eint
120                   reti
121
122
```

Figure 7: Code 4/4 - Part 2

# 3 RESULTS

At the end of the first task, the team is succeeded in displaying given string array on the LCD display(See Figure 1. That was observed written assembly code can interpret end of the line and end of the sequence characters successfully.

At the end of the second task of the experiment, the team is succeeded in generating random numbers and displaying them on the LCD display. To generate random numbers 'Middle Square Weyl Sequence' approach (that was mentioned in the methodology part) are used. End of the implementation, pressing P2.5 button generates a random number between 0 and 128 and shows it on the 16x2 LCD display.

# 4 DISCUSSION

The most important point in the first part was how to interpret the end of the new line and end of the sequence. The team is succeeded in that controlling every character of the string array, and to get whether it is '0dh(new line)' or '00h(end of the sequence)' or regular character. If it is new line character the program executes newline subroutine. This subroutine moves cursor to the beginning of the second line. End of this subroutine program is continues to controlling and processing other characters on the string. If the character is end of the sequence character the program executes endseq subroutine. This subroutine stops the controlling characters entering endless stop cycle. If the character is regular character, the ASCII character are send to LCD display as binary. First of all lower nibble of the character and then upper nibble of these 8-bit character are sent to LCD display. Because the LCD is configured to use it 4-bit mode. When the character send to LCD, the program continues controlling other characters and processing them as above.

In the second part of the experiment, to respond button press interrupt service routine are used from old experiments. Also, an algorithm to generate random numbers are given to the team before the experiment. The team easily implemented this algorithm that mentioned in the method part in detail. Also, the subroutines for the arithmetic operations are used from old experiments.

# 5 CONCLUSION

As always, the team has successfully completed all tasks. At the end of the experiment, the team has learnt manipulating and displaying somethings on the 16x2 LCD Display via using MSP430G2553. Also, gains more experiences how use interrupts to respond

button press.

The team is completed all experiments successfully. All the experiments helped to be more confident how assembly code are written and how MSP430G2553 microcontroller used.

# REFERENCES

[1] Texas Instruments. Msp430 education board document. 2009.

[2] Texas Instruments. Msp430 architecture. 2009.

[3] Texas Instruments. Msp430 architecture. December 2004 - Revised July 2013.

[4] Overleaf documentation https://tr.overleaf.com/learn.

[5] Detailed info on writing reports https://projects.ncsu.edu/labwrite/res/res-studntintro-labparts.html.