# ECE 375 LAB 4

Data Manipulation & the LCD

**Lab Time: Friday 16:00 ~ 17:50**

Hyunjae Kim

# INTRODUCTION

The purpose of this lab is to access 8-bit data in program memory, and then manipulate data to print the data in LCD. By using logical shift of the address of declared 8-bit data, it is able to access the data in program memory, and load the data to the register. After loading data to the register, the program writes the data in the register to LCD. The program manipulates data in program memory, and then write data to LCD on AVR board.

# PROGRAM OVERVIEW

## INITIALIZATION ROUTINE

First, 8-bit data are declared and defined in program memory. Since program memory is containing 16-bit data, the entire program has to access 8-bit data by 8-bit data in one address of program memory. In order to access 8-bit data in program memory, logical shift will be used. Then the shifted address will be loaded to X-register. By these processes, defined 8-bit data is able to be accessed by the program. Thus, initialization routine is complete.

## MAIN ROUTINE

The main routine of the program is keep executing the loop. In the loop, the program does different functions depending on the inputs from port D. If PD0 is pushed, then the program will write defined data to LCD, and if PD1 is pushed, then the program will write same content to LCD in reversed order when PD0 is pushed. Finally, if PD7 is pushed, the program will clear the LCD. The program will keep run infinite loop by doing these three functions.

## SUBROUTINES

1. Write Data on LCD

When PD0 is pushed, the program goes to FUNC0 which writes data on LCD. First, the program access to the address of the first 8-bit data in program memory, and then load it to the register. After loading the register, the program stores the data in the register to the address of LCD. The accessing and manipulating process of second data is same as the first data string. This process is keep iterating until the program confront the address which is set to be the end of string. After this process, the program writes the data on LCD, and then returns to the main of the program.

2. Write Data on LCD – Reversed

When PD1 is pushed, the program goes to FUNC0 which writes data on LCD. First, the program access to the address of the second 8-bit data in program memory, and then load it to the register. After loading the register, the program stores the data in the register to the address of LCD. The accessing and manipulating process of the first data is same as the second data string. This process is keep iterating until the program confront the address which is set to be the end of string. After this process, the program writes the data on LCD, and then returns to the main of the program.

3. Clear LCD

Unlike step 1 or 2, this step does not call a separate function in the program. When PD7 is pushed, the program goes to CLEAR routine which is coded in the program, and then calls the clearing function which is in include file in the program. Then LCD will be cleared. After LCD is cleared, the program returns to the main routine of the program.

## STUDY QUESTIONS

1. In this lab, you were required to move data between two memory types: program memory and data memory. Explain the intended uses and key differences of these two memory types.

The intended use of program memory is to store 16-bit instruction which executes program. The intended use of data memory is to store operand which size is 8-bit, and to use GPRs, and I/O registers.

2. You also learned how to make function calls. Explain how making a function call works (including its connection to the stack), and explain why a RET instruction must be used to return from a function.

Since function calls are subroutine call of AVR assembly, every function call needs return address to execute next instruction in the main routine after the execution of subroutine is completed. Moreover, the return address is pushed in stack when the subroutine is called. In order to pop the return address from the stack, and then jump to the main routine, RET instruction is needed. Thus, RET instruction must be used to return from a function.

3. To help you understand why the stack pointer is important, comment out the stack pointer initialization at the beginning of your program, and then try running the program on your mega128 board and also in the simulator. What behavior do you observe when the stack pointer is never initialized? In detail, explain what happens (or no longer happens) and why it happens.

When the stack pointer is never initialized, the program did not write the data to LCD on AVR board. Moreover, PD0, PD1, and PD7 did not its own function when they were pushed.

The reason is that the program uses jumps and subroutine calls in order to manipulate data and write data to LCD. Jumps and subroutine calls goes to target address after storing return address to the stack. Since the stack pointer is not initialized, jumps and subroutine calls cannot store the return address to the stack. In other words, jumps and subroutine calls are going to the target address, and cannot return to the main routine of the program. As the program cannot return to the main routine, the program stays in the end of the subroutine, and this leads to the event which the program does nothing or actions different from the expectation of the program.

## DIFFICULTIES

Unlike handling contents from data memory, loading data from the program memory was hard to understand and implement LCD as expected. After reviewing the concept, and syntax of handling data in program memory, implementation of LCD worked same as expectation before the implementation. Thus, by reviewing the concept of structure of AVR and syntax, the difficulty was solved.

## CONCLUSION

In LCD, the data defined in program memory was printed. In the process of printing data in LCD, the data going to be printed were defined in the program memory. By using logical shift, each declared data was accessed by X, Y, or Z register, and loaded to other general purpose register, and then printed to LCD. Furthermore, by adding some features to the printing function, AVR board was able to print different data depending on the input from buttons. Therefore, through these processes, AVR board printed different data contents on LCD depending on different inputs – PD0, PD1, PD7.

## SOURCE CODE

```
;***********************************************************
;*       This is the skeleton file for Lab 4 of ECE 375
;*
;*        Author: Hyunjae Kim
;*          Date: 01/28/2022
;*
;***********************************************************

.include "m128def.inc"                  ; Include definition file



;***********************************************************
;*       Internal Register Definitions and Constants
;***********************************************************
.def    mpr = r16                               ; Multipurpose register is required for LCD
Driver

.equ    pd_0 = 0 ;PD0
.equ    pd_1 = 1 ;PD1
.equ    pd_5 = 5 ;PD5
.equ    pd_6 = 6 ;PD6
.equ    pd_7 = 7 ;PD7

;***********************************************************
;*       Start of Code Segment
;***********************************************************
.cseg                                                   ; Beginning of code segment

;***********************************************************
;*       Interrupt Vectors
;***********************************************************
.org    $0000                                   ; Beginning of IVs
                rjmp INIT                               ; Reset interrupt

.org    $0046                                   ; End of Interrupt Vectors

;***********************************************************
;*       Program Initialization
;***********************************************************
INIT:                                                   ; The initialization routine
                 ldi mpr, LOW(RAMEND)  ; Initialize Stack Pointer
                 out SPL, mpr
                 ldi mpr, HIGH(RAMEND)
                 out SPH, mpr

                 ;Initalize port D for input
                ldi             mpr, $00        ; Set Port D Data Direction Register
                out             DDRD, mpr               ; for input
                ldi             mpr, $FF        ; Initialize Port D Data Register
                out             PORTD, mpr

                rcall LCDInit ; Initialize LCD Display, Don't know why this is defined as an
error

                ; NOTE that there is no RET or RJMP from INIT,
                ; this is because the next instruction executed is the
                ; first instruction of the main program

;***********************************************************
;*       Main Program
;***********************************************************
MAIN:

        in mpr, PIND; get pd inputs from port D
        andi mpr, (1 << pd_0 | 1 << pd_1 |1 << pd_5 |1 << pd_6 | 1 << pd_7);

        cpi mpr, (1 << pd_0 | 1 << pd_1 |1 << pd_5|1 << pd_6) ;  check pd7 is pushed
```

```
        breq CLEAR; if pushed, clear the lcd


        cpi mpr, (1 << pd_1 | 1 << pd_7 |1 << pd_5 | 1 << pd_6) ; check pd 0 is pushed
        brne NEXT ; if not check if it is another input
        rcall FUNC0 ; write string_beg and string_end on lcd
        rjmp MAIN; continute with the program

NEXT:
        cpi mpr, (1 << pd_0 | 1 << pd_7 | 1 << pd_5 | 1 << pd_6) ; check pd1 is pushed
        brne MAIN ;if not go to the main
        rcall FUNC1 ;act same as FUNC0, but in reverse order
        rjmp MAIN; continue with the program

CLEAR:
        rcall LCDClr ; clear lcd
        rjmp MAIN ; go back to main after clearing the lcd







;***********************************************************
;*      Functions and Subroutines
;***********************************************************

;-----------------------------------------------------------
; Func0: Write to LCD
; Desc:  Write STRING_BEG and STRING_END to LCD
;       Hyunjae Kim
;                Hello World!
;-----------------------------------------------------------
FUNC0:
        ldi ZL, low(STRING_BEG << 1) ;start of STRING_BEG
        ldi ZH, high(STRING_BEG << 1)

        ldi YL, $00 ;start of line 1 of LCD
        ldi YH, $01

        LOOP:
                lpm mpr, Z+
                st Y+, mpr

                cpi ZL, low(STRING_END << 1) ; iterates until Z register points the address of
STRING_END
                brne LOOP
                rcall LCDWrLn1 ;write on line 1 of LCD

                 ;123412341234123
                 ;stack;4

                ldi YL, $10 ;start of line 2 of LCD
                ldi YH, $01

        LOOP2:
                lpm mpr, Z+
                st Y+, mpr

                cpi ZL, low(END_of_STRING << 1);iterates until the Z register points the address
of END_of_String
                brne LOOP2
                rcall LCDWrLn2   ;write on line 2 of LCD


                ; Execute the function here

                ; Restore variables by popping them from the stack,
```

```
                ; in reverse order
                ret                                             ; End a function with RET\

;-----------------------------------------------------------
; Func1: Write to LCD
; Desc:  Write STRING_BEG and STRING_END to LCD
;       Hello World!
;                Hyunjae Kim
;-----------------------------------------------------------
FUNC1:
        ldi ZL, low(STRING_END << 1) ; Starts with STRING_END
        ldi ZH, high(STRING_END << 1)

        ldi YL, $00 ; starts with first line
        ldi YH, $01

        LOOP3:
                lpm mpr, Z+
                st Y+, mpr

                cpi ZL, low(END_of_STRING << 1) ; iterate until the Z register points the
address of END_of_STRING
                brne LOOP3
                rcall LCDWrLn1; Write on line 1 of LCD

                ldi ZL, low(STRING_BEG << 1) ; Points the start of STRING_BEG
                ldi ZH, high(STRING_BEG << 1)

                ldi YL, $10 ; starts with second line
                ldi YH, $01

        LOOP4:
                lpm mpr, Z+
                st Y+, mpr

                cpi ZL, low(STRING_END << 1) ; iterates until Z register points the address of
STRING_END
                brne LOOP4
                rcall LCDWrLn2 ; Write on line 2 of LCD
        ret             ; go back to main



;***********************************************************
;*      Stored Program Data
;***********************************************************

;-----------------------------------------------------------
; An example of storing a string. Note the labels before and
; after the .DB directive; these can help to access the data
;-----------------------------------------------------------
STRING_BEG:
.DB             "Hyunjae Kim "          ; Declaring data in ProgMem
STRING_END:
.DB             "Hello World!"
END_of_STRING:


;***********************************************************
;*      Additional Program Includes
;***********************************************************
.include "LCDDriver.asm"         ; Include the LCD Driver
```