ECE 375 LAB 4

Data Manipulation & the LCD

Lab Time: Wednesday 10-12

Tu Lam

INTRODUCTION

The purpose of the fourth lab is to print LCD message and learn to pass program memory into the data memory. The lab also introduces data manipulation in AVR assembly, interact with pre-written library, learn to use indirect addressing to access the X, Y, and Z register.

PROGRAM OVERVIEW

The LCD program will run on pre-build asm file that will control how a message is write, clear, and convert to ASCII base the user input into the program. The actual program will include the pre-written asm file into the main asm file as a way to access it function. The function/subroutine will be call from this main asm to control how the message will display on the LCD screen.

Besides the standard MAIN routines within the program, there will be an INIT initialize stage where we will pass any data in the main stage into the register via the route of indirect addressing. The program also looking at setting up the PIND to accept the PD0, PD1, and PD7 to perform different task. This can be range from printing the entire message, reverse it, or clear the message from the LCD screen.

INITIALIZATION ROUTINE

The initialization routine provides a one-time initialization of key registers that allow the LCD program to execute correctly. First, we will tell the PIND to be an input for PD 0, PD 1, PD 7 to perform a certain task in the program. While at that, we will need to initialize the LDCDriver.asm file to help run the subroutine from this file. In this stage, we will initialize data transfer from the program memory to data memory using the indirect way to get into the X, Y, and Z register. Also, learning to initialize stack pointer and learn how to point to the write index in .DB directive.

MAIN ROUTINE

The Main routine executes a simple statement of calling a subroutine in LCDDriver.asm file to write out the two string that was created using the .DB directive. On top of that, the main routine must accept an input from PIND number 0, 1, and 7 to perform a certain task that will change the way how the LCD will display. The main program will print out the two statements first and checking if the user has entered PD 0 as an input to print out both strings. If not, then it moves into subroutine in the main program to figure out more.

ENTER 1 ROUTINE

The Enter1 routine will look at the input of PIND and see if the PD 1 was trigger or not. First, it will compare the input to the binary placement of PIND 1. If the value matches, it will produce a statement to have the string reverse it displays to display the opposite way on how it was originally displays in the beginning of the program. If not, it will jump to the next routine.

ENTER 7 ROUTINE

The Enter7 routine is identical to the Enter 7 routine, except that it will perform the task a little bit differently. The overall base case for this routine is to check if the input at the placement was trigger. If so, the LCD will be clear from the LCD screen and no message will appear on it. If not, it will return my jumping back into the main program to restart over again, looping it in an infinity loop.

ADDITIONAL 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 usage of program memory and data memory are different in a way that how accessing data through AVR is useful to use. In program memory, the objective of this memory is to have non-volatile data meaning anything store in it will be permanent and will be not be lost. For data memory, data can still be store but temporary as data will be lost if something wrong happen to the machine. The intended between these two is to get data from a permanent place to use it temporary to run the program and won't be losing the data that just been use at all.

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.

To make a function call, we could simply use the 'call' opcode or the 'rcall' opcode to help us jump into those function to perform the task that it is needed. The stack pointer helps out during these function call as it help put data information on the stack and we could easily access the data from within a function as we can't pass by argument. The RET instruction must be used to return from a function as it return any changes that was made in the function back out into the main program.

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.

After running the program in the simulator, the result ends up 'undefined' as the machine give out as the output if we commented out on the stack pointer. Without the usage of the stack pointer, it is hard for AVR assembly to run program knowing all the function and subroutine that it need to access right away as stack pointer hold the pointer to the address location to reach to those destination. Without it, it will never be able to perform.

CONCLUSION

For Lab #4, the lab helps to demonstrate on how AVR assembly works and giving us a hand on experience to work around the assembly language. The lab also demonstrates us to practice using stack pointer and learn to ideally use the indirect addressing way to pass data around on the register. Also, learn about the important of data and program memory and how it help to know what useful about them and why is it important in a sense of assembly usage.

SOURCE CODE

Provide a copy of the source code. Here you should use a mono-spaced font and can go down to 8-pt in order to make it fit. Sometimes the conversion from standard ASCII to a word document may mess up the formatting. Make sure to reformate the code so it looks nice and is readable.

```
;***********************************************************
;*
;*      Data Manipulation (Tu_Lam_Lab4_SourceCode.asm)
;*
;*      Description: Learn how data manipulation work in the
;*  Mega128 microcontroller board and use it to display
;*  characters on the LCD.
;*
;*
;***********************************************************
;*
;*       Author: Tu Lam
;*         Date: October 22, 2020
;*
;***********************************************************

.include "m128def.inc"                                  ; Include definition file

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

;***********************************************************
;*      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

                ldi             mpr, (0<<0)|(0<<1)|(0<<7); Set PD0, PD1, PD7 as input
                out             DDRD, mpr               ; Input is set
                ldi             mpr, (1<<0)|(1<<1)|(1<<7); Set up pull-up resistor for PORTD (5V)
                out             PORTD, mpr

                rcall   LCDInit                         ; Initialize LCD Display


        ; Move strings from Program Memory to Data Memory
                ldi             r30, low(STRING1_BEG << 1); Initialize Z-pointer in .DB for string 1 by
taking the address shift to the left
                ldi             r31, high(STRING1_BEG << 1); (multiply by 2) to get to the first index of
the .DB
                ldi             r28, low(LCDLn1Addr)    ; Initialize the character destination (Data
Memory)
                ldi             r29, high(LCDLn1Addr)
                clr             r20
        ; Set r20 to 0
                ldi             r20, 6          ; load r20 with a counter of 6 for 1st string

LOOP1:
                lpm             mpr, Z+         ;Load the string into r16
                st              Y+,     mpr     ;Store content into Y-register
```

```
                dec             r20             ;Decrement the counter
                brne    LOOP1                   ;If zero flag is not equal to 0, keep looping


                ldi             r30, low(STRING2_BEG << 1); Initialize Z-pointer in .DB for string 2
                ldi             r31, high(STRING2_BEG << 1)
                ldi             r28, low(LCDLn2Addr)    ;Initialize the character destination (Data
Memory)
                ldi             r29, high(LCDLn2Addr)
                clr             r20                     ; Set r20 to 0
                ldi             r20, 12                 ; load r20 with a counter of 12 for 2nd string

LOOP2:
                lpm             mpr, Z+                 ;Load the string into r16
                st              Y+,     mpr             ;Store content into Y-register
                dec             r20                     ;Decrement the counter
                brne    LOOP2                           ;If zero flag is not equal to 0, keep looping


                ; 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:                                           ; The Main program
                rcall   LCDWrite
                in              mpr, PIND       ; Get the input of PIND
                cpi             mpr, (1 << 0)   ; Compare if the PD0 is press
                brne    ENTER1                  ; If not equal, jump to PD1
                rcall   LCDWrite                ; Display the strings on the LCD Display
                rjmp    MAIN

ENTER1:
                cpi             mpr, (1 << 1)   ; Compare to PD1
                brne    ENTER7
                rcall   LCDWrLn2
                rcall   LCDWrLn1
                rjmp    MAIN

ENTER7:
                cpi             mpr, (1 << 7)   ; Compare to PD7
                brne    MAIN
                rcall   LCDClr
                rjmp    MAIN                    ; jump back to main and create an infinite

        ; while loop.  Generally, every main program is an

        ; infinite while loop, never let the main program

        ; just run off

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

;-----------------------------------------------------------
; Func: Template function header
; Desc: Cut and paste this and fill in the info at the
;               beginning of your functions
;-----------------------------------------------------------
FUNC:                                           ; Begin a function with a label

        ; Save variables by pushing them to the stack


        ; Execute the function here
```

```
        ; Restore variables by popping them from the stack,
        ; in reverse order
                ret
        ; End a function with RET

;***********************************************************
;*      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
;-----------------------------------------------------------
STRING1_BEG:
.DB             "Tu Lam"                                ; Declaring data in ProgMem
STRING2_BEG:
.DB             "Hello, World"                          ; Declare string no.2
STRING_END:

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