ECE 375 Lab 4

Data Manipulation & the LCD

**Lab Time: Tuesday 8-10**

Zachary DeVita

# Introduction

The purpose of this lab is to interact with the LCD included on the mega128 microcontroller. To accomplish this it was necessary to initialize an assembly program, and move data from the program memory to the data memory. The primary goal of the lab was to get two different strings to display on the LCD screen; one on the top row and the other on the bottom row. After the program was written, it was converted to hex using AVRStudio, and then uploaded to the AVR board using the Universal Programmer.

# Program Overview

The program doesn’t use sensors, it’s taking input, it simply utilizes the LCD screen that is built into the controller to display text. I hard-coded a couple strings into the program which get stored into the program data. These strings are eventually loaded into data memory. Each string utilized a loop to print each character of the string to the LCD screen.

A secondary goal, the challenge, was to get text to scroll across the LCD screen in a marquee-style. This similarly required moving data from the program memory to the data memory, but it also required additional pointers, the use of two loops, and a delay so that the words would be on the screen long enough to read them. The challenge required the string used to similarly be loaded into data memory, as well as, a loop to print each character of the string to the LCD screen, but it then required two additional loops to get the text to scroll across the screen.

Both implementations cannot be run simultaneously which is why the code for the primary objective is commented off in the source code below.

Pre-processor Routine

Prior to the initialization routine is the internal register definitions and constants. This is where the multi-purpose register is defined to register r16, and there are two pointers defined which point to locations in data memory to eventually store the strings for the top and bottom lines of the LCD screen. For the challenge, there are two registers reserved for pointers and a register reserved for the loop counter.

Initialization Routine

The initialization routine provides the initialization of the stack pointer, and initializes the LCD screen. The low byte of the Y register is pointed to the lower byte of the string which is to be displayed on the top line of the LCD screen, while the high byte of the Y register is pointed to the high byte of the same string. The X register is used to point to the string being displayed on the lower line of the LCD screen in a similar fashion.

The strings are then moved from data memory to the program memory using a loop which iterates until the end of each string is reached. This process is repeated for each string used for the primary objective of the lab, as well as, with the challenge.

Main Routine

The Main routine executes the LCDWrite function which essentially concludes the primary objective for the lab. For the challenge, a value is loaded into the i variable to be used for the loop counter. Because ‘Main’ already loops until it is interrupted or the program is closed, it can be utilized as an outer loop. This outer loop calls the LCD write function at its every iteration, as well as, the function being used for the challenge.

The inner loop is simply used to create a brief pause in the program so that the text can be seen. The inner loop continues looping until the variable i=0, and it uses the LCDWait function for each iteration. The reason for the loop is that LCDWait only pauses for one ten-thousandths of a second. Setting i equal to the hex value 64, 100 in decimal, creates a pause for one tenth of a second which can be easily viewed.

Func Routine

The Func routine is where the ‘challenge’ magic happens. The first thing which occurs in this function is that variables are saved by pushing them onto the stack. It is important to return the values of mpr, count, line, and the low and high bytes of the Z register, back to their initial state when a the function is called. So we push these values onto the memory stack so that these variables can later be returned to their original value.

The X, Y & Z registers are used here to load memory addresses pertaining to the string. The two additional pointers I created in the preprocessor routine are used are used here as additional multipurpose registers to load the X, Y & Z values into. After the function has executed, the original for the values for mpr, count, line, and the low and high bytes of the Z register, are popped off the memory stack and restored. Then the function returns back to main part of the program where the function call had left off.

# 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.

Data memory is static random access memory, and it is where variables can be temporarily stored to be used by th program. Values can be read or written to the data memory. Program memory is flash memory, and it is where the program itself is stored. The program memory is non-volatile which means that its contents will not be affected by power loss. Program memory includes things like the application, commands, definitions and constants used by the application. Data memory is volatile, and it contains things like the registers, and it is where runtime variables are stored.

*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.*

A function uses labels to tell a program where the next instruction is. These labels get translated into the actual memory locations by the assembler. If a function is called then the program will jump to the appropriate memory location of the initial instruction for the function and begin executing the commands at that location.

The memory stack is what holds the memory location of where the function was called so that the program knows where to return after the function has executed. This allows that address to be popped off the stack when the function reaches the RET command. Without the RET command the program would simply continue onto the next program memory address, after the function was finished, and would produce undesired results. The program would not know to return back to the address being held by the memory stack.

*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.*

Without initializing the stack the program fails because it is unable to return to the last instruction before the function call. The program knows to return to the address stored in the memory stack, but, since it hasn’t been initialized, the program doesn’t know where to go.

# Conclusion

The requirement for this lab was to write an assembly language program which would output text onto an LED screen. This lab required learning how to write functions, allocate data memory for variables, use loops and branches, and include additional AVR libraries in our program. We utilized X,Y, & Z registers to read in characters from a string constant, and then used the LCDWrite function to output them onto the LCD screen. We wrote the program in AVRStudio loosely using a sample program as a reference, then compiled the code into a .hex file which we were then able to upload to the controller. The result of the program was a success; the LCD screen was able to display a short string on one line as well as my name on the other line. The lab was fun and it allowed me to better understand the implementation of the AVR assembly code. It previously was more cut and paste than this, but with this lab I was actually forced to learn.

# Source Code

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\*

;\* Lab #4 ECE 375

;\*

;\* Learn to use the LCD display on the mega128 microcontroller board.

;\*

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\*

;\*  Author: Zachary DeVita

;\*    Date: 10/18/2016

;\*

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.include "m128def.inc" ; Include definition file

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\* Internal Register Definitions and Constants

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.def mpr = r16 ; Multipurpose register is

; required for LCD Driver

.def ptrL = r23 ; additional register pointer

.def ptrH = r24 ; additional register pointer

.def i = r25 ; loop counter

.equ top\_line = $0100 ; Reserve space for top line of lcd

.equ bottom\_line = $0110 ; Reserve space for bottom line of lcd

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\* 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

rcall LCDInit ; Initialize LCD Display

;rcall LCDClear ; Clear lines of LCD screen

; ldi ZL, low(NAME\_BEG<<1) ; Point Z to the first line of text

; ldi ZH, high(NAME\_BEG<<1)

/\* ldi YL, low(top\_line) ; Point Y to memory where the LCD will be looking

ldi YH, high(top\_line)

ldi XH, high(bottom\_line) ; Point X to memory where the LCD will be looking

ldi XL, low(bottom\_line)\*/

/\*NAME:

lpm mpr, Z+ ; Move strings from Program Memory to Data Memory/post-increment

st Y+, mpr ; Store MPR to X register/post-increment

cpi ZL, low(NAME\_END<<1)     ; Repeat until Z reaches end of line

brne  NAME

cpi ZH, high(NAME\_END<<1)

brne  NAME

STRING:

lpm mpr, Z+ ; Move strings from Program Memory to Data Memory/post-increment

st X+, mpr ; Store MPR to X register/post-increment

cpi ZL, low(STRING\_END<<1)     ; Repeat until Z reaches end of line

brne  STRING

cpi ZH, high(STRING\_END<<1)

brne  STRING\*/

ldi ZL, low(CHALLENGE\_BEG<<1) ; Point Z to the first line of text

ldi ZH, high(CHALLENGE\_BEG<<1)

ldi XL, $00

ldi XH, $01

CHALLENGE:

lpm mpr, Z+ ; Move strings from Program Memory to Data Memory/post-increment

st X+, mpr ; Store mpr in X register and post-increment

cpi ZL, low(CHALLENGE\_END<<1)     ; Repeat until Z reaches end of line

brne  CHALLENGE

cpi ZH, high(CHALLENGE\_END<<1)

brne  CHALLENGE

; 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

; Display the strings on the LCD Display

;rcall LCDWriteByte

rcall LCDWrite

ldi i, $64 ; load 100 into loop counter

WHILE: ; loop creates a pause for 1 second

rcall LCDWait

dec i

brne WHILE

rcall FUNC ; calls function after pause

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

push mpr ; put mpr on memory stack

push count ; put count on memory stack

push line ; put line on memory stack

push ZL ; put ZL on memory stack

push ZH ; put ZH on memory stack

; Execute the function here

; Restore variables by popping them from the stack,

; in reverse order

ldi i, 30 ; set the loop counter

ldi XL, $00 ; load address into X low byte

ldi XH, $01 ; load address into X High byte

ldi YL, $01 ; load address into Y low byte

ldi YH, $01 ; load address into Y High byte

ldi ZL, $1F ; load address into Z low byte

ldi ZH, $01 ; load address into Z High byte

ld ptrH, Z ; load address of Z into into register

ld ptrL, Y ; load address of Y into register

ld mpr, X ; load address of X into multipurpose register

st X+, ptrH ; store ptrH into X register and post increment

st X+, mpr ; store mpr into X register and post increment

mov mpr, ptrL ; move the data held in ptrL register to mpr

ld ptrL, X ; load register X into ptrL

LOOP:

st     X+, mpr ; store the address held in register mpr in X register and post increment

        mov    mpr, ptrL ; move the contents of ptrL register to mpr

        ld     ptrL, X ; load the address held in X register into ptrL register

        dec    I ; decrement the counter

        brne   LOOP : branch: if i!=0 then continue in loop

        ;Pop the values from the stack

        pop line

        pop     count

        pop     ZH

        pop     ZL

        pop     mpr

ret ; End of function

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\* 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

;-----------------------------------------------------------

/\*NAME\_BEG:

.DB "Zachary DeVita" ; Declaring data in ProgMem

NAME\_END:

STRING\_BEG:

.DB "Hello World!" ; Declaring data in ProgMem

STRING\_END:\*/

CHALLENGE\_BEG:

.DB " Zachary DeVita " ; Declaring data in ProgMem

CHALLENGE\_END:

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\* Additional Program Includes

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.include "LCDDriver.asm" ; Include the LCD Driver

.