ECE 375 Lab 4

AVR LCD

Lab Time: Thursday 10 - 12

Josh Matteson

TA Signature

1 Introduction

This lab is dedicated to understanding the LCD for the AVR board. Learning how to initiate, write, and clear the LCD board, as well as move memory between program memeory and data memory. This lab also involves intialization of the stack pointer. The importance of this lab is seen in manipulating data and getting familiar with the stack and it's purpose.

2 Program Overview

The program writes a name and phrase to the LCD, doing this with the LCDDRIVER.asm as a kind of function header file. The phrase "Josh Matteson! " and "Hello World " are written to the board, which is done by first allocating enough memory at the beginning of the program to do so. The majority of the program is moving memory around to be used for later.

3 Initialize Routine

This initialize routine initializes the stack pointer, as well as calling the LCDinit function. This function intializes the the LCD board.

4 Main Program

The main program loads the Z register with the address of the respective text, and then calls "Write" to write either to the first line or second line of the LCD board. This gets the text out of program memory and into data memory. Main is a continuous loop that repeats these steps.

5 Write

The Write function does a number of task. The main purpose is to load and write the phrase into the LCD board. It does the by looping through the data until it reaches 16 characters, and then finally calling LCDWrLnx. This sends the commands to the LCD board and writes the message.

6 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 program memory is the basic storage for everything in the program, general, it is consistent flash memory. Even when the board is turned off, the program memory stays in storage and is available for use. This program memory consist of 64k bits of storage, which consist of 16 bit words.

   On the other end of this, data memory is volatile SRAM. When the board is reset, all the SRAM registers will be reset.

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.

   Functions save certain areas of memory on the stack. The stack pointer is always pointing to the top of the stack, so that when a new function is called, the stack then gets the value of whats currently in the stack pointer pushed on to it. The processor then goes to the location in the operand. The RET command is needed because it will cause the return address to be loaded from the 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 the stack pointer initialized, the program has no return address to jump to. The program then has unpredictable behavior. The LCD board doesn't have anything on it because of this.

7 Difficulties

A minor difficulty was that the LCDDriver.asm didn't have proper indentation with some of the lines. This causes some errors that weren't actually a part of the program I had to write.

8 Conclusion

Overall, the importance of this lab was clearly seen and understood. There is a very important difference between program memory and data memory, so it is critical to understand how these work together and how to move them around. The stack makes it very easy for a program to enter into a subroutine and return to the previous location of the program. It's also useful for pushing variables.

9 Source Code

```
;***********************************************************
;*
;* lcd.asm
;*
;* This program displays a phrase on the LCD board. This is
;*  is done by moving data around, initializing the stack
;*  pointer, and call the LCDDRIVER.asm functions.
;*
;* This is the skeleton file for Lab 4 of ECE 375
;*
;***********************************************************
;*
;*  Author: Josh Matteson
;*    Date: 02/02/2017
;*
;***********************************************************
```

```
.include "m128def.inc" ; Include definition file

;***********************************************************
;* Internal Register Definitions and Constants
;***********************************************************
.def mpr = r16 ; Multipurpose register is
.def Cnt = r23

.cseg ; 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
; Initialize Stack Pointer
LDI R16, LOW(RAMEND) ; Low Byte of End SRAM Address
OUT SPL, R16 ; Write byte to SPL
LDI R16, HIGH(RAMEND) ; High Byte of End SRAM Address
OUT SPH, R16 ; Write byte to SPH

; Initialize LCD Display
rcall LCDInit

; Move strings from Program Memory to Data Memory

; 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

ldi  ZL, low(STRING_BEG<<1) ; Load data of first string from ram into the low byte of Z register
ldi ZH, high(STRING_BEG<<1) ; Load data of first string from ram into the high byte of Z register

rcall Write ; Call the "Write" function

ldi ZL, low(STRING_BEG2<<1) ;Load data of first string from ram into the low byte of Z register
ldi ZH, high(STRING_BEG2<<1) ;Load data of first string from ram into the high byte of Z register

rcall Write2nd ; Call the "Write" function

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: Write
; Desc: This functions writes the phrase that is pointed to
; by the Z registera
```

```
;-----------------------------------------------------------
Write: ; Write first
push mpr ; Save variable by pushing them to the stack
push wait
rcall LCDClrLn1
ldi YL, low(LCDLn1Addr) ; load data addresses into Y
ldi YH, high(LCDLn1Addr)

Write_lp: ; Loops through and reads all the data
lpm mpr, Z+ ; Reads the program data
st Y+, mpr ; Puts the data in memory
dec Cnt ; Decrement the count
brne Write_lp ; Continue looping until all data is read
rcall LCDWrLn1 ; Write to line 2 of LCD
pop wait ; pop wait register from stack
pop mpr ; pop mpr from stack
ret

Write2nd:
push mpr
push wait
rcall LCDClrLn2
ldi YL, low(LCDLn2Addr) ;load the location addresses
ldi YH, high(LCDLn2Addr)

Write_lp2: ; Loops through and reads all the data
lpm mpr, Z+ ; Reads the program data
st Y+, mpr ; Puts the data in memory
dec Cnt ; Decrement the count
brne Write_lp2 ; Continue looping until all data is read
rcall LCDWrLn2 ; Write to line 2 of LCD
pop wait ; pop wait register from stack
pop mpr ; pop mpr from stack
ret ; Return 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 "Josh Matteson!  " ; Declaring data in ProgMem
STRING_END:

STRING_BEG2:
.DB "Hello world     "
STRING_END2:

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