ECE 375 LAB 4

Data Manipulation and the LCD Display

Lab Time: Tuesday 4-6

Sonia Camacho

Owen Markley

INTRODUCTION

The purpose of this lab is to get us hands on experience with the AVR board and changing the display. The skeleton code was provided which got us the base of our program where we needed to add in the loops and initializers to get the display working. We were provided with comments in areas that we needed to complete. The key takeaway from this lab was learning how to move program memory to data memory using assembly code. We learned how using the Z and Y registers can act as our pointers and sort of the middle man between the two memory locations. We were instructed to make 2 strings show up on the LCD board and when different buttons were pushed then different actions would occur. We needed to find a way to move the strings from program memory to data memory to properly be displayed when each button was pushed. Overall it was the first lab writing assembly code hands on.

PROGRAM OVERVIEW

In this lab we needed to use the AVR board to display lines of text. In order to do this we needed to initialize the stack pointer and also initialize the LCD display so that we could start displaying text. Then we had to perform the movement from program memory to data memory

INITIALIZATION ROUTINE

The initialization of the stack pointer is the same as was done in lab 3 where we have the pointer pointing to the top of the stack. We then had to call the initialization from the LCDDriver file where this will allow us to use the display screen.

MAIN ROUTINE

Inside of our main function we do all of the work. We do 3 things in our main function. We first start by getting the input value to be able to check what button has been pressed. We then first check to see if the far right button has been pushed and if it has then we have our pointer Y be pointing to the first line and then we load in our first string, then using the method described in the lab 4 slides we will load the program memory from the Z register and then store it where Y is pointing and we will then decrement our i which is what allows us to capture the entire line of the string. We then check to see if the button to the left is pushed and if it is we do what we did when the right button was pressed only this time we are going to loop twice and when we perform our second loop we do not reset our pointer value but we do reset what our Z value is, we do this because we want the first string to now be printed on the bottom line so if you look at the source code you can see on the first line we load in the second string and then we dont move the pointer because now it will be pointing to the 2nd line but now we load in the first string this was when the left button is pushed the order of the strings are switched around. Finally we check to see if the far left button is pushed and if it has been pushed we want to clear the contents of the display and so we used the clear function from the LCDDriver which will wipe out whatever is in program memory and so there will be nothing left on the screen when we push the far left button.

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 program memory is where an application/program is stored. This can sometimes be modified, but in any instance variables are not stored in the program memory. However, a constant can be placed in here.

Data memory is where the variables are read and written to. The difference between the two can be thought of as this. Program memory tells the CPU how to execute, and the Data memory tells the CPU what to execute the program with/on.

Program memory is normally non volatile, whereas data memory is not. Modern PC's have a small amount or program memory included on them.

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.

The CALL or RCALL instruction is used to tell where the beginning of a subroutine is in the Program memory. Once the program counter is informed of the location of this in memory, it can then continue through the rest of the program's code. Using RET, tells the PC to load back the address of the main routine so that it may continue on outside of the subroutine, and finish 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.

Without the stack pointer being initialized, data is not transferred. The LCD display is still initialized, however there is nowhere for the data to go. This is because the stack pointer as we learned is always pointing to the TOS and without that the stack pointer will not know what is next in the stack. As well as when we ran the simulator nothing even happened on the screen.

DIFFICULTIES

Some difficulties that we had were when we had to handle the left button was pushed. We were having a hard time understanding how to load the first string into the second spot. We then had to refer to the lab 4 slides and draw out how we can not reset the pointer location while still loading a new string. We also struggled to understand the loops because assembly code works differently than what we are used to but we caught on pretty fast.

CONCLUSION

In conclusion this lab was a great way for us to get hands on experience with assembly code and visually seeing how it works. Since we were provided with the starter skeleton code it was a lot simpler than if we were writing all the code from the ground up. We were also provided with the LCDDriver code so that made it really nice when we needed to do functions like clear the program memory. I think we sharpenned our assembly skills and learned new concepts like how to call functions between files and how to make loops and work between program and data memory. It is one thing to read about it in the textbook and it is another to actually write the code that goes along. I think this gave us a good understanding of assembly and memory that will prepare us for the coming labs.

SOURCE CODE

```
;***********************************************************
;*
;* sonia_camacho_owen_markley_lab4_sourcecode
;*
;* Enter the description of the program here
;*
;* This is the skeleton file for Lab 4 of ECE 375
;*
;***********************************************************
;*
;*  Author: SONIA CAMACHO OWEN MARKLEY
;*    Date: 01/28/2020
;*
;***********************************************************
.include "m128def.inc"   ; Include definition file
;***********************************************************
;* Internal Register Definitions and Constants
;***********************************************************
.def mpr = r16    ; Multipurpose register is
.def i = r23     ; required for LCD Driver
.DEF J =R25
.equ WskrR = 0    ; Right Whisker Input Bit
.equ WskrL = 1    ; Left Whisker Input Bit
;***********************************************************
;* 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  mpr, low(RAMEND) ; initialize Stack Pointer
  out  SPL, mpr
  ldi  mpr, high(RAMEND)
  out  SPH, mpr
  rcall LCDInit ; Initialize LCD Display

  ; 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 whisker input from Port D
;*************************************************************************************************
**
;checking for the far right button
  cpi  mpr, (0b11111110) ; Check for Right Whisker input (Recall Active Low)
  brne NEXT   ; Continue with next check
  ldi i , $20
   ldi YL, low(LCDLn1Addr)
   ldi YH, high(LCDLn1Addr)
   ldi ZL ,low( STRING1_BEG << 1)
   ldi ZH ,high( STRING1_BEG << 1)
   ;do the loop from the lab slides
  loop:
   lpm r16,Z+
   ST Y+, r16
   dec i
   brne loop
  rcall LCDWrite   ; Call subroutine HitLeft
;*************************************************************************************************
**
;checking for the left button
NEXT: cpi  mpr, (0b11111101)
  brne NEXT1   ; No Whisker input, continue program
  ldi J , $10
  ;load the string 2 to the first line
   ldi YL, low(LCDLn1Addr)
   ldi YH, high(LCDLn1Addr)
   ldi ZL ,low( STRING2_BEG<< 1)
   ldi ZH ,high( STRING2_BEG << 1)
   ; do the loop from the lab slides
  loop2:
   lpm r16,Z+
   ST Y+, r16
   DEC J
   brne loop2
   ;load the frist string but notice we are not reseting the pointer because the pointer will now
be on the second line
   ldi J, $10
   ldi ZL ,low( STRING1_BEG<< 1)
   ldi ZH ,high( STRING1_BEG << 1)
   ;loop again
  loop2pointo:
   lpm r16,Z+
   ST Y+, r16
   DEC J
   brne loop2pointo
   ;now we can write it
  rcall LCDWrite   ; Call subroutine HitLeft
  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
;*************************************************************************************************
**
;checking to see if we are clearing it
NEXT1:
  cpi mpr, (0b01111111)
  brne MAIN   ; No Whisker input, continue program
  rcall LCDClr   ; Call subroutine HitLeft
  rjmp MAIN

;***********************************************************
;* 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  "(: sonia camacho"  ; Declaring data in ProgMemSTRING2_BEG:
STRING1_END:
STRING2_BEG:
.DB  "sonia camacho :0"
STRING2_END:

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