ECE 375 Lab 2

Data Manipulation and the LCD Display

Lab Time: Friday 4-6

Aaron Vaughan

Bradley Heenk

TA Signature

1 Introduction

The purpose of this lab is to get us familiar with the process of writing our own avr assembly code. This lab uses the LCD screen. This means that we have to include the driver file in order to utilize this additional piece of hardware. The initialization of the stack pointer is crucial for the proper operation of this lab. We learn that without this initialization step, the LCD driver package runs in a continuous loop.
The assignment is to display our name and that of our lab partner with included functionality to swap the lines that they are printed on and clear the screen with the use of the buttons connected to PORTD. While coding this project, my lab partner and I learned how to implement for loops, if statements, and became familiarized with the formatting of the avr assembly code itself. The project reinforced our knowledge of data systems and memory management. The process is not too dissimilar from writing in C. It just takes many many more lines to orchestrate the mnemonics and produce working code.

2 Program Overview

The avr assembly code in this lab displays the text that is reserved in data memory (hard-coded by Bradley and I) on the LCD screen. This is accomplished by reserving program memory for the string data in an array of sequentially accessible char values with a head pointer. The data is then loaded in sequentially in a char by char fashion beginning at the address $0100. The LCD is 16 char-wide sections in length. We padded our names with spaces to fill the entire screen. The reading and writing of the char values to the LCD screen are accomplished by post-incrementing an index pointer to access the next memory address that a char needs to be written to. It goes through all the values in the Partner Name array. This repeats for the second partner's name on the second line. There are two buttons that can be pressed to initiate the writing of the strings to the LCD. One button puts name one on line one name two on line two and the other switch then swaps the names and lines. The last button (S7) clears the screen by calling the LCDClr function defined in the LCD driver package. The details of definitions of registers, constants, subroutines and other important details follow in the next sections.

3 Internal Register Definitions and Constants

In this section of code, we set up the required registers and constants. Most of the internal registers are used up in the LCD driver package so we were very restricted as to which registers we had left over to use for the implementation of our algorithm. We set the forward, reverse, and erase values corresponding to the locations of the switches which are active low.

4 Interrupt Vectors

There is one interrupt vector within the skeleton code for this lab. It sets up the initial starting point of our main program. Beginning at memory address $0000 it simply jumps to the initialization routine.

5 Program Initialization

The stack pointer must be initialized to use the stack in the algorithm. This is typically done, as implemented in the our code, to the end of ram. It takes two cycles to perform this because the stack pointer is 16-bits but the register used to communicate with it is only 8-bits wide. Without the use of the stack pointer and use of that type of data structure, there would not be enough registers to perform the operations. This lab uses the LCD display which takes up a lot of the available registers. To use the LCD we must include some drivers and initialize the LCD display. Also, since we want to have a way to interface with the device with I/O we need to initialize PortD as an input.

6 Main Program

The main program section of this lab runs in a loop and continuously checks for inputs from the buttons connected to PORTD. It accomplishes this by first storing the PIND values into mpr. It then compares the value of mpr to the hard coded values of the forward, reverse, and clear switches. If a switch is pressed the value on the PIND

corresponding to that switch will be 1. If the pressed button matches the condition of the case, then it will send the program to that subroutine to execute the algorithm to execute that command. After a function call, we return back out and loop back up to main to start over again. If no buttons are pressed, the main program loops back and starts over.

7 Subroutines

Subroutine name: S1_DISPLAY
Description: A micro-function for PARTNER_WRITE to simplify what is said in main. This function sets everything up to display partner 1 on line 1 and partner 2 on line 2

Subroutine name: S2_DISPLAY
Description: A micro-function for PARTNER_WRITE to simplify what is said in main. This function sets everything up to display partner 2 on line 1 and partner 1 on line 2

Function name: S8_CLEAR;
Description: This function clears the LCD Screen by calling the; LCDClr in the LCDDriver.asm

Subroutine name: PARTNER_WRITE
Description: Writes partner based on the line types This function depends on countin being set before being called to either 1 or a 2 to indicate which partner. The is true with count out which correlates to which line mpr is set to: 1 means line 1, 2 means line 2

8 Stored Program Data

As described in the program overview, the program data is used for a string of 8-bit char values reserved in program memory using the .db operative. We stored two 16-char values that are to be used as an array of chars to implement printing to a screen. This stored program data section is near the end of the code as an avr style convention.

9 Additional Program Includes

As mentioned in many other sections. The LCD requires a specific driver package. It is here, in this section that the file is included.

10 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.

   Program memory is 16-bits and is arranged so that we can access individual bytes using low(address) or high(address). We use it to store large data structures like arrays of chars. It's primary purpose is to store instruction words. In contrast, 8-bit data memory is used for data needed for arithmetic, logical, or other operations. These can be operands or any other useful data used in computations. If more than 8-bits are required, the data is stored sequentially.

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 or subroutine is called by the mnemonic rcall. When rcall is read by the compiler, the first thing that it does is push the current location in program memory onto the stack so that it can get back when the function is finished. Next, it jumps to the location in memory where the function instructions to carry out the command are stored in memory. The subroutine does its thing and then at the bottom there is a return statement. The return statement pops the return location back off the stack and uses it for the next fetch cycle. The result is that we return back out to the line where we called the function or subroutine. This way, the next line in the program can be ran.

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 the program will still compile but it it will not work on the board. I watch what happens in the debugger. When the LCDInit is called, mpr is pushed onto the stack. The consequence of this is that since the stack pointer was not initialized, it had the value of $0000 stored as its location in memory to point to. The problem arises once the push operation is called by the LCD driver package. Push will set the register data onto the stack and decrement itself. This puts the stack pointer past the RAMEND value of $01FF. So the stack pointer is now in a location outside of valid memory addresses and the program gets stuck in an infinate loop within a subroutine fittingly called LCDWait.

11 Difficulties

Understanding how to use the inputs from the push buttons was a bit difficult. Keeping track of all the data structures while implementing the algorithm with a limited number of registers took a little extra help from Youngbin. The extra credit was difficult to get right. We implemented it two different ways before finding the correct definition of marquee scroll in the lab4 handout.

12 Conclusion

The lab was fun. We learned how important the use of data structures and memory management are, as they pertain to programming a microcontroller. We were tasked with implementing an algorithm in avr assembly language that displayed text strings on an LCD display. We coded, compiled, debugged, compiled again, then loaded the program on our atmega128 boards to see it in action. Memory management and proper use of the stack was important.

13 Source Code

```
;***********************************************************
;*
;* Aaron_Vaughan_and_Bradley_Heenk_Lab4_sourcecode.asm
;*
;* This program uses AVR to display strings on the LCD screen
;* S8 clears the screen, S1 Displays Bradley Heenk line one and
;* Aarron Vaughan line 2. The S2 switch flips the order.
;*
;* This is the skeleton file for Lab 4 of ECE 375
;*
;***********************************************************
;*
;* Author: Bradley Heenk and Aaron Vaughan
;* Date: 10/30/2019
;*
;***********************************************************

.include "m128def.inc" ; Include definition file

;***********************************************************
;* Internal Register Definitions and Constants
;***********************************************************
.def mpr = r16               ; Multipurpose register is required for LCD Driver
.def countin = r23           ; Counter value for loops
.def countout = r24          ; Counter value for loops

.equ forward = 0b11111110    ; Setting up the DP0 switch
.equ reverse = 0b11111101    ; Setting up the PD1 switch
.equ erase = 0b01111111      ; Setting up the PD7 switch

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

out DDRD, mpr           ; Set Port D Data Direction Register
ldi mpr, $FF            ; Initialize Port D Data Register
out PORTD, mpr          ; so all Port D inputs are Tri-State

ldi mpr, $00            ; Empyting mpr with zeroes
ldi countin, $00        ; Empyting countin with zeroes
ldi countout, $00       ; Empyting countout with zeroes

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            ; Record the current values of PIND onto mpr
cpi mpr, forward        ; Check to see if forward button is pressed
brne CASE1              ; If not check the next case
rcall S1_DISPLAY        ; If equal we call S1_DISPLAY
rjmp MAIN               ; Start over jump to MAIN
```

```
CASE1:
cpi mpr, reverse        ; Check to see if the reverse button is pressed
brne CASE2              ; If not check next case
rcall S2_DISPLAY        ; If equal we call S2_DISPLAY
rjmp MAIN               ; Start over jump to MAIN
CASE2:
cpi mpr, erase          ; Check to see if the erase button is pressed
brne MAIN               ; If not jump to MAIN
rcall S8_CLEAR          ; If equal we call S8_CLEAR
rjmp MAIN               ; Start over jump to MAIN


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

;-----------------------------------------------------------
; Sub: S1 Display
; Desc: A micro-function for PARTNER_WRITE to simplify
; what is said in main. This fucntion sets everything up
; to display partner 1 on line 1 and partner 2 on line 2
;-----------------------------------------------------------
S1_DISPLAY:
push countin            ; Pushes countin onto the stack
push countout           ; Pushes countout onto the stack
push mpr                ; Pushes mpr onto the stack

ldi countin, $02        ; Declare partner 1 to prep our fucntion
ldi countout, $02       ; Declare line 1 to prep our fucntion
rcall PARTNER_WRITE     ; Calls PARTNER_WRITE with setup parameters

ldi countin, $01        ; Declare partner 2 to prep our fucntion
ldi countout, $01       ; Declare line 2 to prep our fucntion
rcall PARTNER_WRITE     ; Calls PARTNER_WRITE with setup parameters

pop mpr                 ; Pops mpr off the stack
pop countout            ; Pops countout off the stack
pop countin             ; Pops countin off the stack
ret
```

```
;-----------------------------------------------------------------
; Sub: S2 Display
; Desc: A micro-function for PARTNER_WRITE to simplify
; what is said in main. This fucntion sets everything up
; to display partner 2 on line 1 and partner 1 on line 2
;-----------------------------------------------------------------
S2_DISPLAY:
push countin              ; Pushes countin onto the stack
push countout             ; Pushes countout onto the stack
push mpr                  ; Pushes mpr onto the stack

ldi countin, $01           ; Declare partner 1 to prep our fucntion
ldi countout, $02          ; Declare line 2 to prep our fucntion
rcall PARTNER_WRITE        ; Calls PARTNER_WRITE with setup parameters

ldi countin, $02           ; Declare partner 2 to prep our fucntion
ldi countout, $01          ; Declare line 1 to prep our fucntion
rcall PARTNER_WRITE        ; Calls PARTNER_WRITE with setup parameters

pop mpr                    ; Pops countin off the stack
pop countout               ; Pops countout off the stack
pop countin                ; Pops mpr off the stack
ret

;-----------------------------------------------------------
; Func: S8 Clear
; Desc: This fucntion clears the LCD Screen by calling the
; LCDClr in the LCDDriver.asm
;-----------------------------------------------------------
S8_CLEAR:
rcall LCDClr               ; Calls the LCDClear function to clear display

ret

;-----------------------------------------------------------------
; Sub: Writes partner based on the line types
; Desc: This fucntion depends on countin being set
; before being called to either 1 or a 2 to indicate which
; partner. The is true with count out correlating to which
; line mpr is set to; 1 means line 1, 2 means line 2.
```

```
;-----------------------------------------------------------

PARTNER_WRITE:
cpi countin, $01                ; Checks if we're partner 1
breq PARTNER1                   ; If true branches to the right Partner1
cpi countin, $02                ; Checks if we're partner 2
breq PARTNER2                   ; If true branches to the right Partner2
rjmp PARTNER_EXIT               ; Countin wasn't setup correctly jump to PARTNER_EXIT
PARTNER1:
ldi ZL, low(PARTNER1_BEG<<1)    ; Load left-shifted PARTNER1_BEG low byte into ZL
ldi ZH, high(PARTNER1_BEG<<1)   ; Load left-shifted PARTNER1_BEG high byte into ZH
rjmp CHECKLINES                 ; Jump to check which line we're on
PARTNER2:
ldi ZL, low(PARTNER2_BEG<<1)    ; Load left-shifted PARTNER2_BEG low byte into ZL
ldi ZH, high(PARTNER2_BEG<<1)   ; Load left-shifted PARTNER2_BEG high byte into ZH
CHECKLINES:
cpi countout, $01               ; Checks if countout is a 1 for line 1
breq LINE1                      ; If true branches to LINE1
cpi countout, $02               ; Check if countout is a 2 for line 2
breq LINE2                      ; If true branches to LINE2
rjmp PARTNER_EXIT               ; Countout wasn't setup correctly
                                ; jump to PARTNER_EXIT
LINE1:
ldi YL, low(LCDLn1Addr)         ; Load the LCDLn1Addr into YL low byte
                                ; which points to the beggining of line1
ldi YH, high(LCDLn1Addr)        ; Load the LCDLn1Addr into YH high byte
                                ; which points to the beggining of line1
rjmp STARTWRITING               ; Jump to STARTWRITING
LINE2:
ldi YL, low(LCDLn2Addr)         ; Load the LCDLn2Addr into YL low byte which points
                                ; to the beggining of line2
ldi YH, high(LCDLn2Addr)        ; Load the LCDLn2Addr into YH high byte which points
                                ; to the beggining of line2
STARTWRITING:
ldi countin, $0F                ; Set out countin counter to start at 16
WRITELINES:
lpm mpr, Z+                     ; Load program memory from Z into mpr and post-inc Z
st Y+, mpr                      ; Store mpr into Y and post-inc Y
dec countin                     ; Decrement our countin
brne WRITELINES                 ; If not equal to 0 start loop again
```

```
cpi countout, $01               ; Checks if we wrote to line 1
breq LCDWR1                     ; If true branch to LCDWR1 fucntion
cpi countout, $02               ; Checks if we wrote to line 2
breq LCDWR2                     ; If true branch to LCDWR2 fucntion
rjmp PARTNER_EXIT               ; Countout wasn't setup correctly
                                ; jump to PARTNER_EXIT
LCDWR1:
rcall LCDWrLn1                  ; Call the LCDWrLn1 fucntion to write line 1
rjmp PARTNER_EXIT               ; Jump to PARTNER_EXIT
LCDWR2:
rcall LCDWrLn2                  ; Call the LCDWrLn2 fucntion to write line 2
PARTNER_EXIT:
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
;-----------------------------------------------------------
PARTNER1_BEG:
.DB "BRADLEY HEENK   "          ; Declaring data in ProgMemory
PARTNER1_END:

PARTNER2_BEG:
.DB "AARON VAUGHAN   "          ; Declaring data in ProgMemory
PARTNER2_END:

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

14 Challenge Code

```
    ;***********************************************************
;*
;* Bradley_Heenk_Aaron_Vaughan_Lab4_challenge.asm
;*
;* This program uses AVR to display strings on the LCD screen
;* S8 clears the screen, S1 Displays Bradley Heenk line one and
;* Aarron Vaughan line 2. The S2 switch flips the order.
;*
;* This is the skeleton file for Lab 4 of ECE 375
;*
;***********************************************************
;*
;* Author: Bradley Heenk and Aaron Vaughan
;* Date: 10/31/2019
;*
;***********************************************************

.include "m128def.inc" ; Include definition file

;***********************************************************
;* Internal Register Definitions and Constants
;***********************************************************
.def mpr = r16 ; Multipurpose register is
; required for LCD Driver
.def countin = r23 ; Counter value for loops
.def countout = r24 ; Counter value for loops
.def countwait = r25 ; Counter value for loops

.equ waittime = 25 ; Setup wait time for delays
.equ s1_btn = 0b11111110 ; Setting up the S1
.equ s2_btn = 0b11111101 ; Setting up the S2
.equ s6_btn = 0b11011111 ; Setting up the S8
.equ s7_btn = 0b10111111 ; Setting up the S7
.equ s8_btn = 0b01111111 ; Setting up the S8

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

out DDRD, mpr ; Set Port D Data Direction Register
ldi mpr, $FF ; Initialize Port D Data Register
out PORTD, mpr ; so all Port D inputs are Tri-State

ldi mpr, $00 ; Empyting mpr with zeroes
ldi countin, $00 ; Empyting countin with zeroes
ldi countout, $00 ; Empyting countout with zeroes
ldi countwait, $00 ; Empyting countwait with zeroes

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 ; Setup PIND for input
```

```
cpi mpr, s1_btn ; Compare our first pin to mpr
brne CASE1 ; If not check the next case
rcall S1_INIT ; If equal we call S1_DISPLAY
rjmp MAIN ; Start over jump to MAIN
CASE1:
cpi mpr, s2_btn ; Compare reverse pin to mpr
brne CASE2 ; If not check next case
rcall S2_INIT ; If equal we call S2_DISPLAY
rjmp MAIN ; Start over jump to MAIN
CASE2:
cpi mpr, s6_btn ; Compare reverse pin to mpr
brne CASE3 ; If not check next case
rcall S6_INIT ; If equal we call S2_DISPLAY
rjmp MAIN ; Start over jump to MAIN
CASE3:
cpi mpr, s7_btn ; Compare reverse pin to mpr
brne CASE4 ; If not check next case
rcall S7_INIT ; If equal we call S2_DISPLAY
rjmp MAIN ; Start over jump to MAIN
CASE4:
cpi mpr, s8_btn ; Compare erase pin to mpr
brne MAIN ; If not jump to MAIN
rcall S8_CLEAR ; If equal we call S8_CLEAR
rjmp MAIN ; Start over jump to MAIN


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

;-----------------------------------------------------------
; Sub: S6 Display
; Desc: Our main fucntion to have our text scroll from
; left to right using PD5 where the first line
; starts scrolling and as characters are cut off
; they start scrolling from left to right on line 2.
;       For line 2 is also scrolls left to right and as
; characters are cut off starts scrolling from left
; to right on line 1 until the names have been swapped.
;-----------------------------------------------------------
```

```
S6_DISPLAY:
ldi mpr, $00 ; Initalize MPR to $00
ldi countwait, $01 ; Initalize countwait to $01
ldi countout, $0F ; Initalize countout to $0F
ldi countin, $0F ; Initalize countin to $0F

rcall S1_INIT ; Calls the S1_INIT function
rcall DELAY ; Calls the DELAY function

S1_RELOAD:
ldi countout, $11 ; Load $11 into countout
ldi YL, low(LCDLn1Addr) ; Load the low byte of LCDLn1Addr into YL
ldi YH, high(LCDLn1Addr) ; Load the high byte of LCDLn1Addr into YH
mov countin, countwait ; Move the value of countwait into countin
S1_L1_SPACELOOP:
push mpr ; Push mpr onto the stack
ldi ZL, low(PARTNER2_BEG<<1) ; Load the low byte of PARTNER1_BEG into the ZL
ldi ZH, high(PARTNER2_BEG<<1) ; Load the high byte of PARTNER1_BEG into the ZH
ldi mpr, $10 ; Load the value $10 into mpr
sub mpr, countin ; Subtract countin from mpr
add ZL,mpr ; Add mpr into ZL
lpm mpr, Z+ ; Load program memory from Z into mpr and post-inc Z
st Y+, mpr ; Store mpr into Y and post increment Y
pop mpr ; Pop mpr off the stack

dec countin ; Decrement count in by one
brne S1_L1_SPACELOOP ; If not equal to zero branch to S1_L1_SPACELOOP

ldi ZL, low(PARTNER1_BEG<<1) ; Load the low byte of PARTNER1_BEG into ZL
ldi ZH, high(PARTNER1_BEG<<1) ; Load the high byte of PARTNER1_BEG into Zh

ldi countin, $10 ; Load $10 into countin
sub countin, countwait ; Subtract countwait from countin

S1_L1_WORDLOOP:
push mpr ; Push mpr to the stack
lpm mpr, Z+ ; Load program memory of Z into mpr and post-inc Z
st Y+, mpr ; Store mpr into Y and post increment Y
pop mpr ; Pop mpr off the stack
```

```
dec countin ; Decrement countin by one
brne S1_L1_WORDLOOP ; If not zero branch to S1_L1_WORDLOOP

ldi countout, $11 ; Load $11 into countout
ldi YL, low(LCDLn2Addr) ; Load the low byte of LCDLn2Addr into YL
ldi YH, high(LCDLn2Addr) ; Load the high byte of LCDLn2Addr into YH
mov countin, countwait ; Move the value of countwait into countin

rjmp S1_L2_SPACELOOP ; Jump to S1_L2_SPACELOOP
UN_JUMP: ; This function was used to allow for large jumps
; that require more than 8 bits of memory
rjmp S1_RELOAD ; Jump to S1_RELOAD

S1_L2_SPACELOOP:
push mpr ; Push mpr onto the stack
ldi ZL, low(PARTNER1_BEG<<1) ; Load the low byte of PARTNER1_BEG into ZL
ldi ZH, high(PARTNER1_BEG<<1) ; Load the high byte of PARTNER1_BEG into ZH
ldi mpr, $10 ; Load $10 into mpr
sub mpr, countin ; Subtract countin from mpr
add ZL,mpr ; Add mpr to ZL
lpm mpr, Z+ ; Load from program memory Z into mpr and post-inc Z
st Y+, mpr ; Store mpr into Y and post-inc Y
pop mpr ; Pop mpr off the stack

dec countin ; Decrement countin by one
brne S1_L2_SPACELOOP ; If not zero branch to S1_L2_SPACELOOP

ldi ZL, low(PARTNER2_BEG<<1) ; Load low byte of PARTNER2_BEG into ZL
ldi ZH, high(PARTNER2_BEG<<1) ; Load high byte of PARTNER2_BEG into ZH

ldi countin, $10 ; Load $10 into countin
sub countin, countwait ; Subtract countwait from countin

S1_L2_WORDLOOP:
push mpr ; Push mpr onto the stack
lpm mpr, Z+ ; Load program memory from Z into mpr and post-inc Z
st Y+, mpr ; Store mpr into Y and post-inc Y
pop mpr ; Pop mpr off the stack

dec countin ; Decrement countin by one
```

```
brne S1_L2_WORDLOOP ; If not zero branch to S1_L2_WORDLOOP

inc countwait ; Increment countwait by one

rcall DELAY ; Call our DELAY function
rcall LCDWrite ; Call LCDWrite function

cpi countwait, $00 ; Compare countwait to zero
breq ALMOSTDONE ; If equal to zero branch to ALSMOSTDONE
sub countout, countwait ; Subtract countwait from countin
brne UN_JUMP ; If not equal to zero branch to UN_JUMP

ALMOSTDONE:
rcall DELAY ; Call the DELAY fucntion
rcall S2_INIT ; Call the S2_INIT function

ret

;-----------------------------------------------------------------
; Sub: S7 Display
; Desc: Our main fucntion to have our text scroll from
; right to left using PD6 where the first line
; starts scrolling and as characters are cut off
; they start scrolling from right to left on line 2.
;       For line 2 is also scrolls right to left and as
; characters are cut off starts scrolling from right
; to left on line 1 until the names have been swapped.
;-----------------------------------------------------------------
S7_DISPLAY:
ldi mpr, $00 ; Initalize MPR to $00
ldi countwait, $10 ; Initalize countwait to $01
ldi countout, $0F ; Initalize countout to $0F
ldi countin, $0F ; Initalize countin to $0F

rcall S2_INIT ; Calls the S1_INIT function
rcall DELAY ; Calls the DELAY function

S2_RELOAD:
ldi countout, $11 ; Load $11 into countout
ldi YL, low(LCDLn1Addr) ; Load the low byte of LCDLn1Addr into YL
```

```
ldi YH, high(LCDLn1Addr) ; Load the high byte of LCDLn1Addr into YH
mov countin, countwait ; Move the value of countwait into countin
S2_L1_SPACELOOP:
push mpr ; Push mpr onto the stack
ldi ZL, low(PARTNER2_BEG<<1) ; Load the low byte of PARTNER2_BEG into the ZL
ldi ZH, high(PARTNER2_BEG<<1) ; Load the high byte of PARTNER2_BEG into the ZH
ldi mpr, $10 ; Load the value $10 into mpr
sub mpr, countin ; Subtract countin from mpr
add ZL,mpr ; Add mpr into ZL
lpm mpr, Z+ ; Load program memory from Z into mpr and post-inc Z
st Y+, mpr ; Store mpr into Y and post increment Y
pop mpr ; Pop mpr off the stack

dec countin ; Decrement count in by one
brne S2_L1_SPACELOOP ; If not equal to zero branch to S2_L1_SPACELOOP

ldi ZL, low(PARTNER1_BEG<<1) ; Load the low byte of PARTNER1_BEG into ZL
ldi ZH, high(PARTNER1_BEG<<1) ; Load the high byte of PARTNER1_BEG into Zh

ldi countin, $10 ; Load $10 into countin
sub countin, countwait ; Subtract countwait from countin

S2_L1_WORDLOOP:
push mpr ; Push mpr to the stack
lpm mpr, Z+ ; Load program memory of Z into mpr and post-inc Z
st Y+, mpr ; Store mpr into Y and post increment Y
pop mpr ; Pop mpr off the stack

dec countin ; Decrement countin by one
brne S2_L1_WORDLOOP ; If not zero branch to S2_L1_WORDLOOP

ldi countout, $11 ; Load $11 into countout
ldi YL, low(LCDLn2Addr) ; Load the low byte of LCDLn2Addr into YL
ldi YH, high(LCDLn2Addr) ; Load the high byte of LCDLn2Addr into YH
mov countin, countwait ; Move the value of countwait into countin

rjmp S2_L2_SPACELOOP ; Jump to S2_L2_SPACELOOP
UN_JUMP_2: ; This function was used to allow for large jumps
; that require more than 8 bits of memory
rjmp S2_RELOAD ; Jump to S2_RELOAD
```

```
S2_L2_SPACELOOP:
push mpr ; Push mpr onto the stack
ldi ZL, low(PARTNER1_BEG<<1) ; Load the low byte of PARTNER1_BEG into ZL
ldi ZH, high(PARTNER1_BEG<<1) ; Load the high byte of PARTNER1_BEG into ZH
ldi mpr, $10 ; Load $10 into mpr
sub mpr, countin ; Subtract countin from mpr
add ZL,mpr ; Add mpr to ZL
lpm mpr, Z+ ; Load from program memory Z into mpr and post-inc Z
st Y+, mpr ; Store mpr into Y and post-inc Y
pop mpr ; Pop mpr off the stack

dec countin ; Decrement countin by one
brne S2_L2_SPACELOOP ; If not zero branch to S2_L2_SPACELOOP

ldi ZL, low(PARTNER2_BEG<<1) ; Load low byte of PARTNER2_BEG into ZL
ldi ZH, high(PARTNER2_BEG<<1) ; Load high byte of PARTNER2_BEG into ZH

ldi countin, $10 ; Load $10 into countin
sub countin, countwait ; Subtract countwait from countin

S2_L2_WORDLOOP:
push mpr ; Push mpr onto the stack
lpm mpr, Z+ ; Load program memory from Z into mpr and post-inc Z
st Y+, mpr ; Store mpr into Y and post-inc Y
pop mpr ; Pop mpr off the stack

dec countin ; Decrement countin by one
brne S2_L2_WORDLOOP ; If not zero branch to S2_L2_WORDLOOP

dec countwait ; Increment countwait by one

rcall DELAY ; Call our DELAY function
rcall LCDWrite ; Call LCDWrite function

cpi countwait, $00 ; Compare countwait to zero
breq ALMOSTDONE_2 ; If equal to zero branch to ALSMOSTDONE_2
sub countout, countwait ; Subtract countwait from countin
brne UN_JUMP_2 ; If not equal to zero branch to UN_JUMP_2
```

```
ALMOSTDONE_2:
rcall DELAY ; Call the DELAY function
rcall S1_INIT ; Call the S2_INIT function
ret

;-----------------------------------------------------------------
; Sub: S1 Init
; Desc: A micro-function for PARTNER_WRITE to simplify
; what is said in main. This fucntion sets everything up
; to display partner 1 on line 1 and partner 2 on line 2
;-----------------------------------------------------------------
S1_INIT:
push countin ; Pushes countin onto the stack
push countout ; Pushes countout onto the stack
push mpr ; Pushes mpr onto the stack

rcall LCDClrLn2 ; Calls LCD_ClrLn1 function

ldi countin, $02 ; Declare partner 1 to prep our fucntion
ldi countout, $02 ; Declare line 1 to prep our fucntion
rcall PARTNER_WRITE ; Calls PARTNER_WRITE with setup parameters

rcall LCDClrLn1 ; Calls LCD_ClrLn2 function

ldi countin, $01 ; Declare partner 2 to prep our fucntion
ldi countout, $01 ; Declare line 2 to prep our fucntion
rcall PARTNER_WRITE ; Calls PARTNER_WRITE with setup parameters

pop mpr ; Pops countin off the stack
pop countout ; Pops countout off the stack
pop countin ; Pops mpr off the stack
ret

;-----------------------------------------------------------------
; Sub: S2 Init
; Desc: A micro-function for PARTNER_WRITE to simplify
; what is said in main. This fucntion sets everything up
; to display partner 2 on line 1 and partner 1 on line 2
;-----------------------------------------------------------------
S2_INIT:
```

```
push countin ; Pushes countin onto the stack
push countout ; Pushes countout onto the stack
push mpr ; Pushes mpr onto the stack

rcall LCDClrLn2 ; Calls LCD_ClrLn1 function

ldi countin, $01 ; Declare partner 1 to prep our fucntion
ldi countout, $02 ; Declare line 2 to prep our fucntion
rcall PARTNER_WRITE ; Calls PARTNER_WRITE with setup parameters

rcall LCDClrLn1 ; Calls LCD_ClrLn2 function

ldi countin, $02 ; Declar partner 2 to prep our fucntion
ldi countout, $01 ; Declare line 1 to prep our fucntion
rcall PARTNER_WRITE ; Calls PARTNER_WRITE with setup parameters

pop mpr ; Pops countin off the stack
pop countout ; Pops countout off the stack
pop countin ; Pops mpr off the stack
ret

;-----------------------------------------------------------------
; Sub: S6 Init
; Desc: A micro-function for S6_DISPLAY to simplify
; what is said in main.
;-----------------------------------------------------------------
S6_INIT:
rcall S6_DISPLAY ; Calls S6_DISPLAY
ret

;-----------------------------------------------------------------
; Sub: S7 Init
; Desc: A micro-function for S7_DISPLAY to simplify
; what is said in main.
;-----------------------------------------------------------------
S7_INIT:
rcall S7_DISPLAY ; Calls S6_DISPLAY
ret

;-----------------------------------------------------------
```

```
; Func: S8 Clear
; Desc: This fucntion clears the LCD Screen by calling the
; LCDClr in the LCDDriver.asm
;-----------------------------------------------------------
S8_CLEAR:
rcall LCDClr ; Calls the LCDClear function to clear display
ret

;-----------------------------------------------------------------
; Sub: Delay
; Desc: This micro-fucntion takes care of storing our countwait
; register so we can jsut call this main without having to worry
;-----------------------------------------------------------------
DELAY:
push countwait ; Pushes countwait to the stack
ldi countwait, waittime ; Load our waittime into countwait
rcall TIMER ; Call our TIMER function
pop countwait ; Pop countwait off the stack
ret

;-----------------------------------------------------------------
; Sub: Wait Function
; Desc: A wait loop that is 16 + 159975*waitcnt cycles or roughly
; waitcnt*10ms.  Just initialize wait for the specific amount
; of time in 10ms intervals. Here is the general eqaution
; for the number of clock cycles in the wait loop:
; ((3 * countin + 3) * countout + 3) * countwait + 13 + call
;-----------------------------------------------------------------
TIMER:
push countwait ; Save wait register
push countin ; Save ilcnt register
push countout ; Save olcnt register
push mpr

LOOP: ldi countout, 224 ; load olcnt register
OLOOP: ldi countin, 237 ; load ilcnt register
ILOOP: dec countin ; decrement ilcnt
brne ILOOP ; Continue Inner Loop
dec countout ; decrement olcnt
brne OLOOP ; Continue Outer Loop
```

```
dec countwait ; Decrement wait
brne LOOP ; Continue Wait loop

pop mpr
pop countout ; Restore olcnt register
pop countin ; Restore ilcnt register
pop countwait ; Restore wait register
ret

;-----------------------------------------------------------------
; Sub: Writes partner based on the line types
; Desc: This fucntion depends on countin being set
; before being called to either 1 or a 2 to indicate which
; partner. The is true with count out which corlates to which
; line mpr is set to; 1 means line 1, 2 means line 2.
;-----------------------------------------------------------------
PARTNER_WRITE:
cpi countin, $01 ; Checks if we're partner 1
breq PARTNER1 ; If true branches to the right Partner1
cpi countin, $02 ; Checks if we're partner 2
breq PARTNER2 ; If true branches to the right Partner2
rjmp PARTNER_EXIT ; Countin wasn't setup correctly jump to PARTNER_EXIT
PARTNER1:
ldi ZL, low(PARTNER1_BEG<<1) ; Load left-shited PARTNER1_BEG low byte into ZL
ldi ZH, high(PARTNER1_BEG<<1) ; Load left-shited PARTNER1_BEG high byte into ZH
rjmp CHECKLINES ; Jump to check which line we're on
PARTNER2:
ldi ZL, low(PARTNER2_BEG<<1) ; Load left-shited PARTNER2_BEG low byte into ZL
ldi ZH, high(PARTNER2_BEG<<1) ; Load left-shited PARTNER2_BEG high byte into ZH
CHECKLINES:
cpi countout, $01 ; Checks if countout is a 1 for line 1
breq LINE1 ; If true branches to LINE1
cpi countout, $02 ; Check if countout is a 2 for line 2
breq LINE2 ; If true branches to LINE2
rjmp PARTNER_EXIT ; Countout wasn't setup correctly jump to PARTNER_EXIT
LINE1:
ldi YL, low(LCDLn1Addr) ; Load the LCDLn1Addr into YL low byte which points
; to the beggining of line1
ldi YH, high(LCDLn1Addr) ; Load the LCDLn1Addr into YH high byte which points
; to the beggining of line1
```

```
rjmp STARTWRITING ; Jump to STARTWRITING
LINE2:
ldi YL, low(LCDLn2Addr) ; Load the LCDLn2Addr into YL low byte which points
; to the beggining of line2
ldi YH, high(LCDLn2Addr) ; Load the LCDLn2Addr into YH high byte which points
; to the beggining of line2
STARTWRITING:
ldi countin, $0F ; Set out countin counter to start at 16
WRITELINES:
lpm mpr, Z+ ; Load program memory from Z into mpr and post-inc Z
st Y+, mpr ; Store mpr into Y and post-inc Y
dec countin ; Decrement our countin
brne WRITELINES ; If not equal to 0 start loop again

cpi countout, $01 ; Checks if we wrote to line 1
breq LCDWR1 ; If true branch to LCDWR1 fucntion
cpi countout, $02 ; Checks if we wrote to line 2
breq LCDWR2 ; If true branch to LCDWR2 fucntion
rjmp PARTNER_EXIT ; Countout wasn't setup correctly jump to PARTNER_EXIT
LCDWR1:
rcall LCDWrLn1 ; Call the LCDWrLn1 fucntion to write line 1
rjmp PARTNER_EXIT ; Jump to PARTNER_EXIT
LCDWR2:
rcall LCDWrLn2 ; Call the LCDWrLn2 fucntion to write line 2
PARTNER_EXIT:
ret

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

;***********************************************************
;* 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
;-----------------------------------------------------------
```

```
PARTNER1_BEG:
.DB "BRADLEY HEENK   " ; Declaring data in ProgMem
PARTNER1_END:

PARTNER2_BEG:
.DB "AARON VAUGHAN   " ; Declaring data in ProgMem
PARTNER2_END:
```