ECE 375 Lab 3

Data Manipulation & the LCD

Lab Time: Friday 2 PM - 3:50 PM

Student 1: Winnie Woo
Student 2: Joseph Borisch

TA Signature

1 Introduction

The purpose of the third lab was to learn how to initialize and display characters on the LCD display using data manipulation and learning how to set up the stack pointer, and initializing registers. As well as using the X, Y, Z pointers to perform indirect addressing and declaring constants in the program memory using the .DB directive and moving it through the data memory.

2 Program Overview

This section provides an overview of how the assembly program reacts to button input. The LCD display is initially blank, if PD4 is pressed: the contents of the display is cleared, if PD5 is pressed: the 2 strings are displayed on the first and second line of the LCD. If PD6 is pressed: the contents on the first and second line are swapped.
Besides the standard INIT and MAIN routines, additional functions were created. The ClearButton clears the LCD display, the WriteButton displays the first and second lines onto the LCD and the SwitchButton swaps the first and second lines.

3 Initialization Routine

The initialization routine provides a one-time initialization of the stack pointer so the proper function and subroutines can be called as well as the LCD display.

4 Main Routine

The Main routine starts off by moving the strings declared in program memory to data memory. Then displaying the strings on the LCD display.

5 ClearButton

This function handles the functionality of the LCD display when the clear button is triggered. It saves the program state and clears the LCD display.

6 WriteButton

This function handles the functionality of writing data to the LCD display. The function begins by pulling program memory into data memory so that it can be used by the LCD. This program memory is placed into the Y register at a location in memory where the LCD knows to look for the first and second lines of the display.

7 SwitchButton

This function handles the functionality of switching the lines on the LCD display once the names have been been displayed. This function is nearly identical to WriteButton, but the memory

locations pointed to are switched. Instead of looking for the first line in the appropriate memory location, it looks for line 1 where line 2 would be and vice versa.

8 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 implemented using non-volatile flash memory and retains its contents even after power is turned off. Data memory is used to store bytes and data that's altered by the assembly code. The key difference between the two is that program memory is word addressed while data memory is byte addressed.

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.

   Function calls are implemented as high level languages. It starts from the top of the function and run the code within it and then return back to where the function was called. For the RET instruction, the return address is pushed back onto the stack and the code within the function is executed and once it hits the RET, it pops the address off the stack and returns back to the return address on 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 mega32U4 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.

   If the stack pointer initialization is commented out, then the subroutines and functions no longer work because the return address was never pushed onto the stack.

9 Difficulties

When trying to compile the code, we ran into this error

```
syntax error, unexpected '\n'
```

We had trouble figuring out what was causing this error, and referred to stack overflow and Google. We originally thought to step through the register, we had to step back but that was not the case.

10 Conclusion

In conclusion, the original implementation of the scrolling functionality proved too tricky for us to attempt. However, it was interesting to learn how to manipulate characters on the LCD display.

11 Source Code

```
;
;
;
;***********************************************************
;* This is the skeleton file for Lab 3 of ECE 375
;*
;*  Author: Joseph Borisch and Winnie Woo
;*    Date: 10/14/2022
;*
;***********************************************************

.include "m32U4def.inc"    ; Include definition file

;***********************************************************
;* Internal Register Definitions and Constants
;***********************************************************

.def  mpr = r16           ; Multipurpose register is required for LCD Driver
.equ  clrBut = 4          ; Clear screen input bit
.equ  wrtBut = 5       ; write to screen Input Bit
.equ  swtBut = 6       ; swap lines Input Bit

;***********************************************************
;* Start of Code Segment
;***********************************************************
.cseg                  ; Beginning of code segment

;***********************************************************
;* Interrupt Vectors
;***********************************************************
.org  $0000            ; Beginning of IVs
      rjmp INIT           ; Reset interrupt

.org  $0056            ; End of Interrupt Vectors

;***********************************************************
;* Program Initialization
;***********************************************************
INIT:                  ; The initialization routine
   ; Initialize Stack Pointer
      ldi      mpr, low(RAMEND)
      out      SPL, mpr   ; Load SPL with low byte of RAMEND
      ldi      mpr, high(RAMEND)
      out      SPH, mpr   ; Load SPH with high byte of RAMEND
   ; Initialize LCD Display
      rcall LCDInit
```

```
   ;define variables for strings to be displayed by LCD

   ; Initialize Port D for input
      ldi     mpr, $00   ; Set Port D Data Direction Register
      out     DDRD, mpr     ; for input
      ldi     mpr, $FF   ; Initialize Port D Data Register
      out     PORTD, mpr    ; so all Port D inputs are Tri-State


   ; 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
;***********************************************************
     rcall lcdclr            ;clear screen
     rcall lcdbacklighton    ;backlight on
MAIN:                      ; The Main program
     in   mpr, PIND          ;get button input
     com     mpr               ;acive low buttons
     andi  mpr, (1<<clrBut|1<<swtBut|1<<wrtBut) ;logic and to find what button is
        pressed
     cpi     mpr, (1<<clrBut)   ;check for clear button input
     brne  NEXT1
     rcall ClearButton       ;call subroutine ClearButton
     rjmp  MAIN             ;continue with program
NEXT1:  cpi     mpr, (1<<swtBut)   ;check for switch button
     brne  NEXT2
     rcall SwitchButton      ;call subroutine SWithcButton
     rjmp  MAIN
NEXT2:  cpi     mpr, (1<<wrtBut)   ;check for write button
     brne  MAIN             ;no input, continue program
     rcall WriteButton       ;call subroutine WriteButton

     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: ClearButton
; Desc: Handles functionality of the LCD display when the clear button
;is triggered
;-----------------------------------------------------------
ClearButton:  ; Begin a function with a label
```

```
      ; Save variables by pushing them to the stack
      push  mpr
      in    mpr, SREG     ;save program state
      push  mpr
      ; Execute the function here
      rcall LCDclr        ;clear the lcd screen
      ; Restore variables by popping them from the stack,
      ; in reverse order
      pop     mpr
      out     SREG, mpr
      pop     mpr
      ret        ; End a function with RET

;-----------------------------------------------------------
; Func: Write Button
; Desc: Handles functionality of the LCD display when the write button
;is triggered
;-----------------------------------------------------------
WriteButton:  ; Begin a function with a label
      ; Save variables by pushing them to the stack
      push  mpr
      in    mpr, SREG     ;save program state
      push  mpr
      ; Execute the function here

      ldi ZL, low(STRING_BEG<<1)  ;load lo string 1 into Z
      ldi ZH, high(STRING_BEG<<1)   ;load high string 1 into Z
      ldi YL, $00             ;initialize address pointing to first line of low byte
         on LCD
      ldi YH, $01             ;initialize address pointing to first line of high byte
         on LCD
LOOP1:  lpm mpr, Z+              ;mpr gets lo byte of z
      st Y+, mpr              ;low Y gets low z
      cpi ZL, low(STRING_BEG<<1) ; compare the value of the low_end after shifting one
         bit, required for printing an entire string and not just first letter
      brne  LOOP1
      ldi ZL, low(STRING_END<<1)  ;load lo string 2 into Z
      ldi ZH, high(STRING_END<<1)   ;load high string 2 into Z
      ldi   YL, $10               ;initialize address pointing to second line of low
         byte on LCD
      ldi   YH, $01               ;initialize address pointing to second line of high
         byte on LCD
LOOP2:  lpm mpr, Z+
      st Y+, mpr
      cpi ZL, low(STRING_END<<1) ; compare the value of the low_end after shift one
         bit, if it is reached that, the break
      brne  LOOP2
      rcall LCDwrite
      ; Restore variables by popping them from the stack,
```

```
      ; in reverse order
      pop     mpr
      out     SREG, mpr
      pop     mpr
      ret        ; End a function with RET

;-----------------------------------------------------------
; Func: Switch Button
; Desc: Handles functionality of the LCD display when the switch button
;is triggered
;-----------------------------------------------------------
SwitchButton: ; Begin a function with a label
           ; Save variables by pushing them to the stack
           ;function nearly identical to WriteButton, just switches the memopry
              location that Y is defined by. Memory locations get flipped
      push  mpr
      in    mpr, SREG     ;save program state
      push  mpr
      ; Execute the function here
      ldi ZL, low(STRING_END<<1)  ;load lo string 1 into Z
      ldi ZH, high(STRING_END<<1)   ;load high string 1 into Z
      ldi YL, $00
      ldi YH, $01
LOOP3:  lpm mpr, Z+              ;mpr gets lo byte of z
      st Y+, mpr               ;low Y gets low z
      cpi ZL, low(STRING_END<<1)  ;compare the value of the low_end after shift one
         bit, if it is reached that, the break
      brne  LOOP3
      ldi ZL, low(STRING_BEG<<1)  ;load lo string 2 into Z
      ldi ZH, high(STRING_BEG<<1)   ;load high string 2 into Z
      ldi   YL, $10
      ldi   YH, $01
LOOP4:  lpm mpr, Z+
      st Y+, mpr
      cpi ZL, low(STRING_BEG<<1)
      brne  LOOP4
      rcall LCDwrite
      ; Restore variables by popping them from the stack,
      ; in reverse order
      pop     mpr
      out     SREG, mpr
      pop     mpr
      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
;-----------------------------------------------------------

STRING_BEG:
.DB     "Joe Borisch    "      ; Declaring data in ProgMem
STRING_END:
.DB     "Winnie Woo     "      ; Declaring data in ProgMem


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