ECE 375 LAB 4

Data Manipulation & LCD

Lab Time: Tuesday 4-6

Alexander Uong

INTRODUCTION

The purpose of this lab is to provide us more experience programming in assembly through initializing a program, manipulating data, and interacting with the LCD display on our ATMEGA128 board. We were provided a skeleton code template that provided a basic template that we were asked to edit and modify, adding functionality through tasks such as declaring a stack pointer, initializing the LCD display, moving data from program memory to data memory, and displaying contents onto the LCD display.

PROGRAM OVERVIEW

The program for this lab allows us to transfer strings from program memory to data memory, displaying the data onto the LCD Display of the ATMEGA128 board. In the program, are several tasks that are completed in order to allow the program to function as intended. Prior to running the program, several tasks are completed in the program initialization (INIT); the stack pointer is initialized, the pins on the board are initialized for input, the LCD display is initialized, and the string data from program memory is transferred to data memory.

Within the MAIN routine, the program checks for user input through the pressing of the pins on the board. Based on which pin is pressed, the board will behave differently, shown through the LCD Display. If pin PD0 is pressed, the program will display my name on the first line of the LCD, and the phrase “Hello, World” on the second line of the LCD. If pin PD1 is pressed, the lines are swapped, with “Hello, World” being printed on the first line and my name being printed on the second line of the LCD. If pin PD7 is pressed, the text on the LCD will be cleared.

Besides the INIT and MAIN routines in the program, there are two other additional routines that were created and implemented: SWAPSTRINGS and INITIALIZE. SWAPSTRINGS is to handle the case where if the pin PD1 is pressed, storing the “Hello, World” string into the first line of the LCD Display and my name into the second line of the LCD Display. INITIALIZE is simply what was found in INIT, where the strings are moved from program memory to data memory. This is to easily copy the strings from program memory to data memory, as calling the function LCDClr clears the contents in data memory.

It’s also important to note that the strings are stored in program memory at the bottom of the program. The program “LCDDriver.asm” is also included at the bottom of the program, allowing us to use specific LCD subroutines to display and modify data onto the LCD display.

SWAPSTRINGS ROUTINE

This routine is nearly identical to the copying of data from program memory to data memory in INIT, except the strings swap LCD line addresses. Instead of my name being copied into the first line and “Hello, World” being copied into the second line of the LCD display, they’re swapped, with “Hello, World” being copied onto the first line and my name being copied onto the second line.

INITIALIZE Routine

This routine simply transfers the string data from program memory to data memory. Identical to the initial copying in INIT, this routine was created to easily copy data from program memory to data memory in our MAIN routine. When the routine LCDClr is called, it clears the contents of data memory, so the strings in program memory must be copied to data memory each time LCDClr is called. It also allows consistent behavior of our board when the three pins (PD0, PD1, PD7) are pressed on the board.

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 used for storing and saving a program. Program memory is read-only, when the program executes, for example. AVR only executes programs stored in program memory. Data memory is readable and writeable, consisting of memory such as I/O memory, extended I/O memory, and internal SRAM. A difference between these two memory types is program memory is 16 bits wide, as data memory is only 8 bits wide.

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.

To make a function call, the command “call” or “rcall” is used. Call is a long call to a subroutine, as rcall is a relative call to a subroutine. The RET instruction is used to tell the program when to return from the function call, indicating the end of the function. The address of where the function was called is stored on the stack. The program then returns to where the function was called in the program following the end of the function.

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.

When the stack pointer is never initialized, the program builds successfully but doesn’t show correct behavior, with nothing executing. Like stated above, the addresses of where functions are called are stored on the stack. With no stack being present, the program isn’t able to function properly.

Sources:

http://download.mikroe.com/documents/compilers/mikroc/avr/help/avr_memory_organization.htm#:~:text=Program%20Memory%20(ROM)%20is%20used,keeping%20intermediate%20results%20and%20variables.

CONCLUSION

In this lab, many different areas of computer organization and assembly language were prioritized, with gaining experience in programming in assembly language being the main goal. This lab also provided us greater insights to certain tasks, such as data manipulation, initializing a program, indirect addressing, and how to interact with our board, such as through the LCD display.

SOURCE CODE

```
;
; Lab4.asm
;
; Created: 2/1/2021 11:17:13 PM
; Author : Alex Uong
;

;***********************************************************
;*
;*      main.asm
;*
;*      Transfers strings from program memory to data memory, displaying the contents of the
strings initially
;*      in program memory, onto the LCD Display of the ATMEGA128 Board
;*
;*      This is the skeleton file for Lab 4 of ECE 375
;*
;***********************************************************
;*
;*       Author: Alexander Uong
;*         Date: 2/1/2021
;*
;***********************************************************

.include "m128def.inc"               ; Include definition file


;***********************************************************
;*      Internal Register Definitions and Constants
;***********************************************************
.def    mpr = r16                               ; Multipurpose register is
                                                        ; required for LCD Driver

.def counterRegister = r24           ;multipurpose register to serve as counter


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


                ; Initialize LCD Display
                rcall LCDInit ; Initializes LCD peripheral interface

                ; Move strings from Program Memory to Data Memory

                ldi ZL, LOW(STRING_NAME_BEG << 1); Z register pointer points to low byte of string
data in program memory
                ldi ZH, HIGH(STRING_NAME_BEG << 1); Z register pointer points to high byte of
string data in program memory

                ldi YL, LOW(LCDLn1Addr)                         ; Set Y register to point to low
byte destination of Line 1 of the LCD Display. From LCDDriver
                ldi YH, HIGH(LCDLn1Addr)               ; Set Y register to point to high byte
destination of Line 1 of the LCD Display. From LCDDriver

                ldi counterRegister, 14                                         ;load the constant
value 14 into register r24, with 14 being the # of characters in my name

LINE1:  //This loads the contents of STRING_NAME_BEG one character at a time
                lpm mpr, Z+ ; load data from z register into register r16. Post increment Z to
point to next character in string
                st Y+,mpr    ; store data from r16 into data memory (LCD Display line1). Post
increment to move to next location in data memory
                DEC counterRegister    ; decrement loop counter
                BRNE LINE1     ; branch if not equal, remains in LOOP if counter is still greater
than 0

                //Setting the Z register to point to the hello world string now that the first
string has been loaded into the LCD

                ldi ZL, LOW(HELLOWORLD_BEG << 1); Z register pointer points to lower byte of
string data in program memory
                ldi ZH, HIGH(HELLOWORLD_BEG << 1); Z register pointer points to higher bytes of
string data in program memory

                ldi YL, LOW(LCDLn2Addr)                       ; Set Y register to point to low byte
destination of Line 2 of the LCD Display. From LCDDriver
                ldi YH, HIGH(LCDLn2Addr)              ; Set Y register to point to high byte
destination of Line 2 of the LCD Display. From LCDDriver

                ldi counterRegister, 12                                       ;load the constant
value 12 into register r24, with 12 being the # of characters in the string "Hello, World"

LINE2:
                lpm mpr, Z+ ; load data from z register into register r16. Post increment Z to
point to next character in string
                st Y+,mpr    ; store data from r16 into data memory (LCD Display line2). Post
increment to move to next location in data memory
                DEC counterRegister    ; decrement loop counter
                BRNE LINE2     ; branch if not equal, remains in LOOP if counter is still greater
than 0



                ; 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

               IN mpr,PIND                      ; Get whisker input from PORT D
               andi mpr, (0b11111110|0b11111101 |0b01111111)       //logical and
               CPI     mpr, (0b11111110)       ;Check if PD0 is pressed
               BRNE NEXT                               ;Continue with next check
               rcall INITIALIZE                ;Copy strings through moving the strings from
program to data memory. This is to copy the strings to the LCD after LCDClr
               rcall LCDWrite                  ;Writes contents to the LCD Display
               rjmp  MAIN                              ;Continue with the program
NEXT:

               CPI     mpr, (0b11111101)       ;Check if PD1 is pressed
               BRNE NEXT2                              ;continue with next2 check
               rcall SWAPSTRINGS               ;Calls subroutine swapstrings
               rcall LCDWrite                  ;Writes contents to the LCD Display
               rjmp  MAIN                              ;Continue with the program
NEXT2:
               CPI mpr, (0b01111111)           ;Check if PD7 is pressed
               BRNE MAIN                                       ;Continues back to main
               rcall LCDClr                            ;Clears display
               rjmp MAIN                                       ;Continue with the program



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

;-----------------------------------------------------------
; Func: SWAPSTRINGS
; Desc: Swaps line 1 and line 2 on lcd display by copying the strings in reverse order
;
;-----------------------------------------------------------
SWAPSTRINGS:                                    ; 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
               rcall LCDClr                             ;Clears both lines of data and lines
on LCD display
               ldi ZL, LOW(HELLOWORLD_BEG << 1); Z register pointer points to low byte of string
data in program memory
               ldi ZH, HIGH(HELLOWORLD_BEG << 1); Z register pointer points to high byte of
string data in program memory

               ldi YL, LOW(LCDLn1Addr)                  ; Set Y register to point to low
byte destination of Line 1 of the LCD Display. From LCDDriver
               ldi YH, HIGH(LCDLn1Addr)                ; Set Y register to point to high byte
destination of Line 1 of the LCD Display. From LCDDriver

               ldi counterRegister, 12                  ;load the constant value 12 into
register r24, with 12 being the # of characters in the string "Hello, World"

LINEONE:       //This loads the contents of HELLOWORLD_BEG one character at a time
               lpm mpr, Z+ ; load data from z register into register r16. Post increment Z to
point to next character in string
               st Y+,mpr    ; store data from r16 into data memory (LCD Display line1). Post
increment to move to next location in data memory
               DEC counterRegister    ; decrement loop counter
               BRNE LINEONE   ; branch if not equal, remains in LOOP if counter is still greater
than 0

               //Setting the Z register to point to the hello world string now that the first
string has been loaded into the LCD

               ldi ZL, LOW(STRING_NAME_BEG << 1); Z register pointer points to lower byte of
string data in program memory
```

```
                ldi ZH, HIGH(STRING_NAME_BEG << 1); Z register pointer points to higher bytes of
string data in program memory

                ldi YL, LOW(LCDLn2Addr)                         ; Set Y register to point to low byte
destination of Line 2 of the LCD Display. From LCDDriver
                ldi YH, HIGH(LCDLn2Addr)                ; Set Y register to point to high byte
destination of Line 2 of the LCD Display. From LCDDriver

                ldi counterRegister, 14                         ;load the constant value 14 into
register r24, with  being the # of characters in my name


LINETWO:
                lpm mpr, Z+ ; load data from z register into register r16. Post increment Z to
point to next character in string
                st Y+,mpr    ; store data from r16 into data memory (LCD Display line2). Post
increment to move to next location in data memory
                DEC counterRegister    ; decrement loop counter
                BRNE LINETWO    ; branch if not equal, remains in LOOP if counter is still greater
than 0

                ret                                             ; End a function with RET
;-----------------------------------------------------------------
; Sub:  INITIALIZE
; Desc: Copies the strings from program memory into data memory
;
;-----------------------------------------------------------------

INITIALIZE:
                rcall LCDClr                                     ;Clears both lines of data and lines
on LCD display
                ldi ZL, LOW(STRING_NAME_BEG << 1); Z register pointer points to low byte of string
data in program memory
                ldi ZH, HIGH(STRING_NAME_BEG << 1); Z register pointer points to high byte of
string data in program memory

                ldi YL, LOW(LCDLn1Addr)                          ; Set Y register to point to low
byte destination of Line 1 of the LCD Display. From LCDDriver
                ldi YH, HIGH(LCDLn1Addr)                ; Set Y register to point to high byte
destination of Line 1 of the LCD Display. From LCDDriver

                ldi counterRegister, 14                                 ;load the constant
value 14 into register r24, with 14 being the # of characters in my name

LINE1 1:        //This loads the contents of STRING NAME BEG one character at a time
                lpm mpr, Z+ ; load data from z register into register r16. Post increment Z to
point to next character in string
                st Y+,mpr    ; store data from r16 into data memory (LCD Display line1). Post
increment to move to next location in data memory
                DEC counterRegister    ; decrement loop counter
                BRNE LINE1_1    ; branch if not equal, remains in LOOP if counter is still greater
than 0

                //Setting the Z register to point to the hello world string now that the first
string has been loaded into the LCD

                ldi ZL, LOW(HELLOWORLD_BEG << 1); Z register pointer points to lower byte of
string data in program memory
                ldi ZH, HIGH(HELLOWORLD_BEG << 1); Z register pointer points to higher bytes of
string data in program memory

                ldi YL, LOW(LCDLn2Addr)                         ; Set Y register to point to low byte
destination of Line 2 of the LCD Display. From LCDDriver
                ldi YH, HIGH(LCDLn2Addr)                ; Set Y register to point to high byte
destination of Line 2 of the LCD Display. From LCDDriver

                ldi counterRegister, 12                                 ;load the constant
value 12 into register r24, with 12 being the # of characters in the string "Hello, World"

LINE2_1:
```

```
                lpm mpr, Z+ ; load data from z register into register r16. Post increment Z to
point to next character in string
                st Y+,mpr   ; store data from r16 into data memory (LCD Display line2). Post
increment to move to next location in data memory
                DEC counterRegister    ; decrement loop counter
                BRNE LINE2_1   ; branch if not equal, remains in LOOP if counter is still greater
than 0

                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_NAME_BEG:
.DB             "Alexander Uong"              ; Declaring string name data in ProgMem
STRING_NAME_END:

HELLOWORLD_BEG:
.DB             "Hello, World"          ; Declaring string name data in ProgMem
HELLOWORLD_END:



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