ECE 375 LAB 4

Introduction to AVR Development Tools

Lab Time: Friday 2-4

Hao Truong

INTRODUCTION

The purpose of this lab is to learn to interact with the LCD included in the atMega128 microcontroller board. The LCD is used to print the strings stored in Program Memory. Students will learn how to manipulate data in AVR assembly, how to initialize a program such as initializing Stack Pointer and input port, and how to perform indirect addressing using Y and Z pointers. Students will also learn to use pre-defined library functions.

PROGRAM OVERVIEW

Students will use three buttons such as PD0, PD1, and PD7 to interact with the LCD. When the button 0 (PD0) is pressed, the LCD will print their name on the first line and either the string "Hello, World" or their partner's name on the second line. When button 1 (PD1) is pressed, the LCD will do the opposite as when PD0 is press by swapping the positions of strings. When button 7 (PD7) is triggered, the LCD will clear both lines.

Beside standard INIT and MAIN routines within the program, two additional subroutines were created and used. The ButtonDP0 and ButtonDP1 provide the basic functionality for handling either when PD0 or PD1 is pressed respectively.

Another important thing is that the number of characters stored in the Program Memory must be even, otherwise it will contain some garbage letters. In order to do so, spaces can be added to make the number of letters even.

INITIALIZATION ROUTINE

The initialization routine provides a one-time initialization of key registers that allow the program to execute correctly. First, it is always necessary to set up the Stack Pointer, this allows the proper use of function and subroutine calls. Port D is initialized to inputs and will receive the inputs from PD0, PD1, and PD7. The LCD Display is also initialized by calling the LCDInit subroutine from the included file named LCDDriver.asm so that other subroutines from this file can be used later in the program.

MAIN ROUTINE

The Main routine executes a simple infinite loop that checks to see if any of the button is pressed. This is done by reading the input from Port D and comparing the input with three 8-bit values such as 0b11111110 (meaning PD0 is hit), 0b11111101 (PD1 is hit), and 0b01111111 (PD7 is hit). The main routine then checks to see which button is hit and then calls the subroutine according to that button. Finally, it will call Main routine again to move the program back to the beginning of the routine to repeat the process.

SUBROUTINES

1. ButtonDP0 Routine

The ButtonPD0 routine first moves the first string (my name) from Program Memory to Data Memory. This is accomplished by having Z point to the first character (H in this case) from the string and Y point to the data memory address of character destination ($0100 or Line 1 of LCD). Then the string is transferred character by character to mpr register and then to data memory since there is no direct path from Program Memory to Data

Memory. The same process is repeated to move the second string (Hello, World) from Program Memory to Data Memory except that Y now points to $0110 or Line 2 of LCD. Finally, LCDWrite is called to display both strings on the LCD.

2. ButtonPD1 Routine

This routine is does the same as Button PD0 except that the data memory address of character destinations are swapped so that the LCD displays the “Hello, World” on the first line and my name (Hao Truong) on the second line

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 non-volatile meaning that data stored in stays forever, it will not disappear even when power goes out. Data Memory, on the other hand, can be used to stored data; however, the data is lost when something happens (when power goes out). The intended use of these two memory types is to prevent data loss. Data can be restored after something unexpected occurs by moving data from Program Memory to Data Memory.

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.

Calling a function can be done by using instruction call followed by a name of a function. Stack Pointer is used for storing temporary data, for storing local variables and for storing return addresses after interrupts and subroutine calls. When a function is called, the address of the function and the return address (address of the next instruction to be executed) are pushed to the stack. When the program counters RET, it will pop the return address and load into PC making the next instruction to be executed the instruction after the CALL instruction.

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.*

After commenting out the stack pointer initialization, result was undefined. SP holds the address of the function when it is called. Without the SP, the program does not know where the function is located leading the result to be undefined.

CONCLUSION

In this lab, we were introduced to the basics of data manipulation in AVR assembly, and the important of initializing Stack Pointer. We were able to move data from Program Memory to Data Memory by performing indirect addressing and use pre-written library functions. We also learned how to interact with the LCD included in the atMega128 microcontroller board.

SOURCE CODE

Provide a copy of the source code. Here you should use a mono-spaced font and can go down to 8-pt in order to make it fit. Sometimes the conversion from standard ASCII to a word document may mess up the formatting. Make sure to reformate the code so it looks nice and is readable.

```
;***********************************************************
;*
;*      Hao_Truong_Lab4_sourcecode
;*
;*      Data Manipulation
;*
;*      This is the skeleton file for Lab 4 of ECE 375
;*
;***********************************************************
;*
;*       Author: Hao Truong
;*         Date: 10/29/2021
;*
;***********************************************************

.include "m128def.inc"               ; Include definition file

;***********************************************************
;*      Internal Register Definitions and Constants
;***********************************************************
.def    mpr = r16                            ; Multipurpose register is
                                                            ; required for LCD Driver
.def    count1 = r23


.equ    Button0 = 0
.equ    Button1 = 1
.equ    Button7 = 7


;***********************************************************
;*      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)       ;  Load  the  low  byte  of  ram's  end
addrress\
               out                   SPL, mpr                      ; Set the SP low register
               ldi                   mpr, high(RAMEND)      ;  Load  the  high  byte  of  ram's  end
address
               out                   SPH, mpr                      ; set the SP high register

               ; Initialize LCD Display
               RCALL          LCDInit

               ; 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
Direction 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
;***********************************************************
MAIN:                                                ; The Main program
               ; Main function design is up to you. Below is an example to brainstorm.


               ; Display the strings on the LCD Display
               in                    mpr, PIND                   ; get input from Port D
;              andi          mpr, (1<<Button0 | 1<<Button1 | 1<<Button7)
               cpi                   mpr, (0b11111110)     ; check for input Button0
               brne          Next                          ; continue with next check if not
equal
               rcall         ButtonPD0                     ; call subroutine ButtonDP0
               rjmp          MAIN
Next:
               cpi                   mpr, (0b11111101)     ; check for Button1
               brne          Next1                         ; continue with next check if not
equal
               rcall         LCDClr                        ; clear both lines of LCD
               rcall         ButtonPD1                     ; call subroutine ButtonDP1
               rjmp          MAIN
Next1:
               cpi                   mpr, (0b01111111)     ; check for Button7
               brne          MAIN
               rcall         LCDClr                        ; clear both lines of LCD

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

;-----------------------------------------------------------
; sub: ButtonPD0
; Desc: When you press PD0: Displays your name (Hao Truong) on the first line of the LCD,
;              and a phrase like ”Hello World!” on the second line of the screen.
;-----------------------------------------------------------
ButtonPD0:                                                 ; Begin a function with a label
               ; Save variables by pushing them to the stack
               push   mpr                          ; save mpr register
               push   count1                ; save count1
               push   ZL                           ; save ZL
               push   ZH                           ; save ZH
               push   YL                           ; save YL
               push   YH                           ; save YH

               ; Execute the function here
               ; Move strings from Program Memory to Data Memory
               ; move first string from Program Memory to Data Memory
               ldi           ZL, low(STRING_BEG << 1)            ;   extract   low   byte   of
STRING_BEG
               ldi           ZH, high(STRING_BEG << 1)           ;   extract   high   byte   of
STRING_BEG
                                                                                        ; Z now
points to first char (H)
```

```
               ldi            YL, $00                                          ;   Y   <-   data
memory address of character destination (line 1)
               ldi            YH, $01
               ldi            count1, 10                                       ;    used    to
count down chars, first line contains 10 letters
Line1:
               lpm            mpr, Z+                             ; load Z to mpr
               st             Y+, mpr                             ; load mpr to Y
               dec            count1                              ;  count  down  #  of  words  to
add
               BRNE    Line1                              ; if not done loop

               ; Move second string from Program Memory to Data Memory
               ldi            ZL, low(STRING_END << 1)            ;   extract   low   byte   of
STRING_BEG
               ldi            ZH, high(STRING_END << 1)           ;   extract   high   byte   of
STRING_BEG
                                                                                      ; Z now
points to first char (H)
               ldi            YL, $10                                          ;   Y   <-   data
memory address of character destination (line 2)
               ldi            YH, $01
               clr            count1
               ldi            count1, 12                                       ;    used    to
count down chars, first line contains 12 letters
Line2:
               lpm            mpr, Z+                             ; load Z to mpr
               st             Y+, mpr                             ; load mpr to Y
               dec            count1                              ;  count  down  #  of  words  to
add
               BRNE    Line2                              ; if not done loop

               rcall          LCDWrite                    ; write both lines of the LCD


               ; Restore variables by popping them from the stack,
               ; in reverse order
               pop            YH
               pop            YL
               pop            ZH
               pop            ZL
               pop            count
               pop            mpr

               ret                                        ; End a function with RET


;-----------------------------------------------------------
; sub: ButtonPD0
; Desc: When you press PD0: Displays (Hello, World) on the first line of the LCD,
;              and my name ”Hao Truong” on the second line of the screen.
;-----------------------------------------------------------
ButtonPD1:                                                ; Begin a function with a label
               ; Save variables by pushing them to the stack
               push    mpr                         ; save mpr register
               push    count1               ; save count1
               push    ZL                          ; save ZL
               push    ZH                          ; save ZH
               push    YL                          ; save YL
               push    YH                          ; save YH

               ; Execute the function here
               ; Move strings from Program Memory to Data Memory
               ; move first string from Program Memory to Data Memory
               ldi            ZL, low(STRING_END << 1)            ;   extract   low   byte   of
STRING_BEG
               ldi            ZH, high(STRING_END << 1)           ;   extract   high   byte   of
STRING_BEG
                                                                                      ; Z now
points to first char (H)
```

```
               ldi            YL, $00                                             ;   Y   <-   data
memory address of character destination (line 1)
               ldi            YH, $01
               clr            count1
               ldi            count1, 12                                          ;    used    to
count down chars, first line contains 10 letters
Loop:
               lpm            mpr, Z+                              ; load Z to mpr
               st             Y+, mpr                              ; load mpr to Y
               dec            count1                               ;  count  down  #  of  words  to
add
               BRNE   Loop                                  ; if not done loop

               ; Move second string from Program Memory to Data Memory
               ldi            ZL, low(STRING_BEG << 1)             ;   extract   low   byte   of
STRING_BEG
               ldi            ZH, high(STRING_BEG << 1)            ;   extract   high   byte   of
STRING_BEG
                                                                                          ; Z  now
points to first char (H)
               ldi            YL, $10                                             ;   Y   <-   data
memory address of character destination (line 2)
               ldi            YH, $01
               clr            count1
               ldi            count1, 10                                          ;    used    to
count down chars, first line contains 12 letters
Loop1:
               lpm            mpr, Z+                              ; load Z to mpr
               st             Y+, mpr                              ; load mpr to Y
               dec            count1                               ;  count  down  #  of  words  to
add
               BRNE   Loop1                                 ; if not done loop

               rcall          LCDWrite                      ; write both lines of the LCD


               ; Restore variables by popping them from the stack,
               ; in reverse order
               pop            YH
               pop            YL
               pop            ZH
               pop            ZL
               pop            count
               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            "Hao Truong"          ; Declaring data in ProgMem
STRING_END:
.DB            "Hello, World"

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