ECE 375 Lab 4

Data Manipulation & the LCD

Lab Time: Wednesday 10a-12n

Robert Detjens

David Headrick

TA Signature

1 Introduction

This lab is an introduction to using the LCD Driver library and manipulating data in both program and data memory.

2 Program Overview

This program initializes the chip and LCD, and in response to button presses displays or clears text on the LCD.

In more detail, this program:

- initializes the stack pointer to allow for subroutine calls
- clears the LCD
- copies string constants from program memory into data memory
- listens to button presses and displays different text for different buttons

3 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 read only and is the only location that can contain program code. Data memory is the actual RAM on the MCU and is where all data – i.e. everything that is not instructions – should be located to be used by the program.

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.

   When running the `CALL` instruction, the address of the next instruction after the `CALL` is pushed to the stack. When `RET`tig, it pops that address back off the stack and goes back to that location. If that address is not popped off the stack when returning, it will stay around and cause problems as there is now extra stuff on the stack that probably is not supposed to be there anymore.

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.

If the stack pointer is not correctly /initialized, the first time that subroutine (e.g. `LCDInit`) tries to `RET`, it gets a bogus address from uninitialized memory and faults and restarts the execution from the top.

4 Difficulties

Initially, the stack pointer was incorrectly initialized with the high byte actually getting set to the low byte of the stack address. This was not caught for several minutes and caused much confusion before realizing our simple mistake.

We also were not initially aware that the `LCDClr` proc cleared the data in Data memory, and only thought that it cleared the LCD directly. After realizing this, we then correctly re-copied the strings back into memory after clearing.

5 Conclusion

After exploring this lab, we now feel confident in interacting with data in both memory pools and calling functions either within or externally defined.

6 Source Code

```
;***********************************************************
;*
;*  Robert Detjens & David Headrick -- Lab  4 sourcecode
;*
;*  This code interacts with the LCD and data/program memory.
;*
;***********************************************************
;*
;*   Author: Robert Detjens
;*            David Headrick
;*     Date: 10/22/2021
;*
;***********************************************************

.include "m128def.inc"      ; Include definition file

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

.def   waitcnt = r17    ; wait function param
.def    ilcnt = r18    ; Inner Loop Counter
.def    olcnt = r19    ; Outer Loop Counter
```

```
; button definitions
.equ      BUTT0 = 0b11111110
.equ      BUTT1 = 0b11111101
.equ      BUTT7 = 0b01111111

;***********************************************************
;*  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 the Stack Pointer
    ldi         mpr,low(RAMEND)
    out         SPL, mpr
    ldi         mpr,high(RAMEND)
    out         SPH, mpr

    ; Initialize LCD Display
    rcall   LCDInit

    ; load strings
    rcall   LOAD_STRINGS

    ; setup button inputs
    ; Initialize Port D for button inputs
    ldi         mpr, 0b00000000   ; Set Port D Data Direction Register
    out         DDRD, mpr         ; for input
    ldi         mpr, 0b11111111    ; Initialize Port D Data Register
    out         PORTD, mpr        ; so all Port D inputs are Tri-State

;***********************************************************
;*  Main Program
;***********************************************************
```

```
MAIN:               ; The Main program

    ; read buttons
    in      mpr,    PIND

    cpi     mpr,    BUTT0
    brne    NO0
      ; button 0: "Name \n Hello"
      rcall     LOAD_STRINGS
      rcall     LCDWrLn1
      rcall     LCDWrLn2
    NO0:

    cpi     mpr,    BUTT1
    brne    NO1
      ; button 1: "Hello \n Name"
      rcall     LOAD_STRINGS_SWAPPED
      rcall     LCDWrLn1
      rcall     LCDWrLn2
    NO1:

    cpi     mpr,    BUTT7
    brne    NO7
      ; button 7: clear
      rcall     LCDClr
    NO7:

    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: LOAD_STRINGS
; Desc: Loads both strings from program memory
;-----------------------------------------------------------
LOAD_STRINGS:               ; Begin a function with a label
    ; save old mpr
    push    mpr
```

```
    ; Move strings from Program Memory to Data Memory
    ; location of string in program memory
    ldi     ZL,   low(NAMESTR_S << 1)
    ldi     ZH,   high(NAMESTR_S << 1)
    ; dest addr in data memory (0x0100)
    ldi     YL,   $00
    ldi     YH,   $01
    str1_l:
      lpm     mpr,  Z+
      st      Y+,   mpr
      cpi     YL, low(NAMESTR_E << 1)
      brne    str1_l

    ; String 2
    ldi     ZL,   low(HELLOSTR_S << 1)
    ldi     ZH,   high(HELLOSTR_S << 1)
    ; dest addr in data memory (0x0100)
    ldi     YL,   $10
    ldi     YH,   $01
    str2_l:
      lpm     mpr,  Z+
      st      Y+,   mpr
      cpi     YL, low(HELLOSTR_E << 1)
      brne    str2_l

    ; Restore variables by popping them from the stack,
    ; in reverse order
    pop       mpr

    ret             ; End a function with RET


;-----------------------------------------------------------
; Func: LOAD_STRINGS_SWAPPED
; Desc: Loads both strings from program memory, in reverse order
;-----------------------------------------------------------
LOAD_STRINGS_SWAPPED:               ; Begin a function with a label
    ; save old mpr
    push     mpr

    ; String 2
    ldi     ZL,   low(HELLOSTR_S << 1)
    ldi     ZH,   high(HELLOSTR_S << 1)
    ; dest addr in data memory (0x0100)
```

```
    ldi     YL,   $00
    ldi     YH,   $01
    str2s_l:
      lpm     mpr,  Z+
      st      Y+,   mpr
      cpi     YL, low(HELLOSTR_E << 1)
      brne    str2s_l

    ; String 1
    ldi     ZL,   low(NAMESTR_S << 1)
    ldi     ZH,   high(NAMESTR_S << 1)
    ; dest addr in data memory (0x0110)
    ldi     YL,   $10
    ldi     YH,   $01
    str1s_l:
      lpm     mpr,  Z+
      st      Y+,   mpr
      cpi     YL, low(NAMESTR_E << 1)
      brne    str1s_l


    ; Restore variables by popping them from the stack,
    ; in reverse order
    pop       mpr

    ret             ; End a function with RET


;-----------------------------------------------------------------
; Sub:  Wait
; 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 * ilcnt + 3) * olcnt + 3) * waitcnt + 13 + call
;-----------------------------------------------------------------
WAIT:
  push         waitcnt          ; Save waitregister
  push         ilcnt            ; Save ilcntregister
  push         olcnt            ; Save olcntregister

Loop:   ldi   olcnt, 224        ; load olcnt register
OLoop:  ldi   ilcnt, 237        ; load ilcnt register
ILoop:  dec   ilcnt             ; decrement ilcnt
```

```
  brne        ILoop             ; Continue InnerLoop
  dec         olcnt             ; decrementolcnt
  brne        OLoop             ; Continue OuterLoop
  dec         waitcnt           ; Decrementwait
  brne        Loop              ; Continue Waitloop

  pop         olcnt             ; Restore olcntregister
  pop         ilcnt             ; Restore ilcntregister
  pop         waitcnt           ; Restore waitregister
  ret                           ; Return fromsubroutine

;***********************************************************
;*  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
;-----------------------------------------------------------
NAMESTR_S:
.DB    "Robert & David  "
NAMESTR_E:
HELLOSTR_S:
.DB    "Hello, world!   "
HELLOSTR_E:

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

6.1 Challenge code

```
;***********************************************************
;*
;*  Robert Detjens & David Headrick -- Lab  4 CHALLENGE code`
;*
;*  This code interacts with the LCD and data/program memory.
;*  Also shifts the bytes around on other button presses.
;*
;***********************************************************
;*
;*   Author: Robert Detjens
;*           David Headrick
```

```
;*     Date: 10/22/2021
;*
;***********************************************************

.include "m128def.inc"      ; Include definition file

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

.def   waitcnt = r17     ; wait function param
.def    ilcnt = r18     ; Inner Loop Counter
.def    olcnt = r19     ; Outer Loop Counter


; button definitions
.equ      BUTT0 = 0b11111110
.equ      BUTT1 = 0b11111101
.equ      BUTT5 = 0b11011111
.equ      BUTT6 = 0b10111111
.equ      BUTT7 = 0b01111111

;***********************************************************
;*  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 the Stack Pointer
    ldi         mpr,low(RAMEND)
    out         SPL, mpr
```

```
    ldi         mpr,high(RAMEND)
    out         SPH, mpr

    ; Initialize LCD Display
    rcall   LCDInit

    ; load strings
    rcall   LOAD_STRINGS

    ; setup button inputs
    ; Initialize Port D for button inputs
    ldi         mpr, 0b00000000   ; Set Port D Data Direction Register
    out         DDRD, mpr         ; for input
    ldi         mpr, 0b11111111    ; Initialize Port D Data Register
    out         PORTD, mpr        ; so all Port D inputs are Tri-State

;***********************************************************
;*  Main Program
;***********************************************************
MAIN:               ; The Main program

    ; read buttons
    in     mpr,   PIND

    cpi   mpr,   BUTT0
    brne  NO0
      ; button 0: "Name \n Hello"
      rcall   LOAD_STRINGS
      rcall   LCDWrLn1
      rcall   LCDWrLn2
    NO0:

    cpi   mpr,   BUTT1
    brne  NO1
      ; button 1: "Hello \n Name"
      rcall   LOAD_STRINGS_SWAPPED
      rcall   LCDWrLn1
      rcall   LCDWrLn2
    NO1:

    cpi   mpr,   BUTT5
    brne  NO5
      ; button 5: rotate left
```

```
    ; start at end of strings, 0x011F
    ; X is one ahead
    ldi   XL,   0x20
    ldi   XH,   0x01
    ldi   YL,   0x1F
    ldi   YH,   0x01
    swap_l_loop:
      ; swap end value all the way to the start
      ld    r1,   -X
      ld    r2,   -Y
      st    X,    r2
      st    Y,    r1
      cpi   YL,   0x00
      brne  swap_l_loop

    ; update display
    rcall   LCDWrLn1
    rcall   LCDWrLn2

    ; wait a bit so it onlt moves once
    ldi   waitcnt,    25
    rcall WAIT
  NO5:

  cpi   mpr,   BUTT6
  brne  NO6
    ; button 6: rotate right

    ; start at start of strings, 0x0100
    ; X is one ahead
    ldi   XL,   0x01
    ldi   XH,   0x01
    ldi   YL,   0x00
    ldi   YH,   0x01
    swap_r_loop:
      ; swap end value all the way to the end
      ld    r1,   X
      ld    r2,   Y
      st    X+,    r2
      st    Y+,    r1
      cpi   YL,   0x1F
      brne  swap_r_loop

    ; update display
```

```
      rcall   LCDWrLn1
      rcall   LCDWrLn2

      ; wait a bit so it onlt moves once
      ldi   waitcnt,    25
      rcall WAIT
    NO6:

    cpi   mpr,   BUTT7
    brne  NO7
      ; button 7: clear
      rcall   LCDClr
    NO7:

    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: LOAD_STRINGS
; Desc: Loads both strings from program memory
;-----------------------------------------------------------
LOAD_STRINGS:             ; Begin a function with a label
    ; save old mpr
    push     mpr

    ; Move strings from Program Memory to Data Memory
    ; location of string in program memory
    ldi     ZL,   low(NAMESTR_S << 1)
    ldi     ZH,   high(NAMESTR_S << 1)
    ; dest addr in data memory (0x0100)
    ldi     YL,   $00
    ldi     YH,   $01
    str1_l:
      lpm     mpr,  Z+
      st      Y+,   mpr
      cpi     YL, low(NAMESTR_E << 1)
      brne    str1_l
```

```
    ; String 2
    ldi     ZL,   low(HELLOSTR_S << 1)
    ldi     ZH,   high(HELLOSTR_S << 1)
    ; dest addr in data memory (0x0110)
    ldi     YL,   $10
    ldi     YH,   $01
    str2_l:
      lpm     mpr,  Z+
      st      Y+,   mpr
      cpi     YL, low(HELLOSTR_E << 1)
      brne    str2_l

    ; Restore variables by popping them from the stack,
    ; in reverse order
    pop       mpr

    ret             ; End a function with RET


;-----------------------------------------------------------
; Func: LOAD_STRINGS_SWAPPED
; Desc: Loads both strings from program memory, in reverse order
;-----------------------------------------------------------
LOAD_STRINGS_SWAPPED:               ; Begin a function with a label
    ; save old mpr
    push     mpr

    ; String 2
    ldi     ZL,   low(HELLOSTR_S << 1)
    ldi     ZH,   high(HELLOSTR_S << 1)
    ; dest addr in data memory (0x0100)
    ldi     YL,   $00
    ldi     YH,   $01
    str2s_l:
      lpm     mpr,  Z+
      st      Y+,   mpr
      cpi     YL, low(HELLOSTR_E << 1)
      brne    str2s_l

    ; String 1
    ldi     ZL,   low(NAMESTR_S << 1)
    ldi     ZH,   high(NAMESTR_S << 1)
    ; dest addr in data memory (0x0110)
    ldi     YL,   $10
```

```
    ldi     YH,   $01
    str1s_l:
      lpm     mpr,  Z+
      st      Y+,   mpr
      cpi     YL, low(NAMESTR_E << 1)
      brne    str1s_l


    ; Restore variables by popping them from the stack,
    ; in reverse order
    pop       mpr

    ret             ; End a function with RET


;-----------------------------------------------------------------
; Sub:  Wait
; 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 * ilcnt + 3) * olcnt + 3) * waitcnt + 13 + call
;-----------------------------------------------------------------
WAIT:
  push         waitcnt          ; Save waitregister
  push         ilcnt            ; Save ilcntregister
  push         olcnt            ; Save olcntregister

Loop:   ldi   olcnt, 224        ; load olcnt register
OLoop:  ldi   ilcnt, 237        ; load ilcnt register
ILoop:  dec   ilcnt             ; decrement ilcnt
  brne        ILoop             ; Continue InnerLoop
  dec         olcnt             ; decrementolcnt
  brne        OLoop             ; Continue OuterLoop
  dec         waitcnt           ; Decrementwait
  brne        Loop              ; Continue Waitloop

  pop         olcnt             ; Restore olcntregister
  pop         ilcnt             ; Restore ilcntregister
  pop         waitcnt           ; Restore waitregister
  ret                           ; Return fromsubroutine

;***********************************************************
;*  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
;-----------------------------------------------------------
NAMESTR_S:
.DB    "Robert & David  "
NAMESTR_E:
HELLOSTR_S:
.DB    "Hello, world!   "
HELLOSTR_E:

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