ECE 375 LAB 4

Data Manipulation & the LCD

Lab Time: Wednesday 5-7pm

Rhea Mae Edwards

Jackson Neff

INTRODUCTION

The lab write-up should be done in the style of a professional report/white paper. Proper headers need to be used and written in a clean, professional style. Proof read the report to eliminate both grammatical errors and spelling. The introduction should be a short 1-2 paragraph section discussing what the purpose of this lab is. This is not merely a copy from the lab handout, but rather your own personal opinion about what the object of the lab is and why you are doing it. Basically, consider the objectives for the lab and what you learned and then briefly summarize them. For example, a good introduction to lab 1 may be as follows.

The purpose of this lab is to illustrate how to move data from program memory to data memory. We were tasked with declaring strings in program memory, moving them into a specific 4 bytes of data memory, and then printing it to the LCD screen. The main difficulty is getting the data into the appropriate location in data memory.

PROGRAM OVERVIEW

This section provides an overview of how the assembly program works. Take the time to write this section in a clear and concise manner. You do not have to go into so much detail that you are simply repeating the comments that are within your program, but simply provide an overview of all the major components within your program along with how each of the components work. Discuss each of your functions and subroutines, interesting program features such as data structures, program flows, and variables, and try to avoid nitty-gritty details. For example, simple state that you "First initialized the stack pointer," rather than explaining that you wrote such and such data values to each register. These types of details should be easily found within your source code. Also, do not hesitate to include figures when needed. As they say, a picture is worth a thousand words, and in technical writing, this couldn't be truer. You may spend 2 pages explaining a function which could have been better explained through a simple program-flow chart. As an example, the remainder of this section will provide an overview for the basic BumpBot behavior.

The code only has 2 sections, the INIT function and the MAIN function. The strings to be displayed on the LCD are defined at the bottom of the code, and declared within program memory. The entire point of this lab is to demonstrate the capacity of the AVR to move these declared strings into data memory, where they can be stored. The INIT function does most of the work. In a nutshell, it stores the value of the defined strings, and the address of the location in data memory, and then stores the string to that address. MAIN simply calls LCDwrite, assuming everything has been taken care of in INIT.

INITIALIZATION ROUTINE

The bulk of the lab code takes place within the INIT function. First, we initialize the stack pointer, even though we don't need it. Then we call LCDinit in order to activate the display. The display can be printed to from two specific 16-bit lines of memory, which begin at $0100 and $0110 in the data memory. In order to get our strings into these lines, we load the Z register with a string, and the Y register with the address $0100. Then we run a loop that copies the contents of the Z register into register mpr and then from there into the address pointed to by Y. We do the low byte of Z first, then the high byte. We then repeat the same process for the second string.

MAIN ROUTINE

MAIN doesn't do much. It infinitely loops on itself and calls LCDWrite each loop, which just maintains the message printed on the screen indefinitely.

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.

Data memory is meant to be used for storage. Reading and writing to and from data memory is very common. Program memory is where the instructions in the code itself are stored. It is not meant to store variables, although it can store constants. The key difference is that variables in data memory can be manipulated and stored, whereas constants defined in program memory can be manipulated, but not stored.

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 call works by storing the current program status (relevant registers, PC, and SR) on the stack. Then, the program flow jumps to the beginning of the called function and continues from there until the end of the function. At the end of the function, values need to be popped off the stack and back into their registers. The last value to be popped is the address in program memory from which the function was called. RET brings the program flow back to that address.

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.

Nothing happens now. We don't see anything on the LCD display or anything in the simulator window. The reason for this is that before we write any values, we call LCDInit. When this is called, it should perform a function and then return to where it was called from. However, we don't have the return address in the stack pointer anymore, so the machine doesn't know how to get back.

CONCLUSION

In the end, the lab was an effective tool for showing how to move data to different parts of memory. It was neat to see these primitive microcontrollers print customized text to the screen. This lab took longer than normal and has been the toughest so far. One thing that confused us was bit shifting, and how that factored in.

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.

```
;***********************************************************
;*      File Name:
;*      main.asm
;*      ece375-L4 skeleton.asm
;*
;*      Enter the description of the program here
```

```
;*
;*  Prints out two strings on the LCD Display on the AVR Board
;*
;*      This is the skeleton file for Lab 4 of ECE 375
;*
;***********************************************************
;*
;*       Author: Rhea Mae Edwards, Jackson Neff
;*         Date: Feburary 7th, 2017
;*
;***********************************************************

.include "m128def.inc"               ; Include definition file

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

;***********************************************************
;*      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)
               OUT   SPL, R16
               LDI   R16, HIGH(RAMEND)
               OUT   SPH, R16

               ; Initialize LCD Display
               rcall   LCDInit

               ; Move strings from Program Memory to Data Memory
                      LDI ZL, low(NAME_STRING_BEG<<1)
                      LDI ZH, high(NAME_STRING_BEG<<1)
                      LDI YL, low(LCDLn1Addr)
                      LDI YH, high(LCDLn1Addr)

               loop:
                      LPM            mpr, Z+
                      st             Y+, mpr
                      CPI            ZL, low(NAME_STRING_END<<1)
                      brne   loop
                      cpi            ZH, high(NAME_STRING_END<<1)

                      LDI    ZL, low(HELLO_STRING_BEG<<1)
                      LDI    ZH, high(HELLO_STRING_END<<1)
                      LDI    YL, low(LCDLn2Addr)
                      LDI    YH, high(LCDLn2Addr)

               loop2:
                      LPM            mpr, Z+
                      st             Y+, mpr
                      CPI            ZL, low(HELLO_STRING_END<<1)
                      brne   loop2
                      cpi            ZH, high(HELLO_STRING_END<<1)
```

```
              ; 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
              rcall LCDWrite                ; Display the strings on the LCD Display

              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: Template function header
; Desc: Cut and paste this and fill in the info at the
;              beginning of your functions
;-----------------------------------------------------------
FUNC:                                                 ; 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

              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
;-----------------------------------------------------------
NAME_STRING_BEG:
.DB           "Jackson Neff "               ; Declaring data in ProgMem
NAME_STRING_END:
HELLO_STRING_BEG:
.DB           "Hello World "        ; Declaring data in ProgMem
HELLO_STRING_END:

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