ECE 375 LAB 3

Data Manipulation & the LCD

Lab session: 010

Time: 1200-1350

Author name: Levi Stidham

Programming partner name: N/A

INTRODUCTION

The purpose of this lab exercise is to gain an understanding of how to manipulate data in AVR, as well as learn how to initialize a program that uses stack pointers, different registers, and peripheral devices. We will be doing this by setting up the LCD screens on our microcontrollers, programming a message into the microcontroller that will be displayed on the LCD. Learning how to use various pointers is important because that will allow us to perform indirect addressing. This lab will also allow us to gain familiarity with constant data in program memory and how to move between program memory and data memory, as well as using prewritten libraries. All of these concepts put together will create a message on our LCD that has a different message on the top and bottom lines as well as the ability to scroll between the two lines marquee style.

DESIGN

Pseudocode:

Call LCD init

Call LCD Clr

Call LCD Backlight on

Initialize the stack

Initialize PORT D and DDRD for input

Initialize PORTB and DDRB for output

Set all buttons to high value (0b11110000) because they are active low when pressed

Check to see if D4,D5 or D7 is pressed.

If D4 is pressed, clear the LCD data, if D5 is pressed display the stored message, if D7 is pressed display the stored messages in a scrolling marquee style.

D5:

Clear the lcd

Put the string stored in the appropriate Z register (split high/low)

Point the Y register at the byte location of the LCD Driver.

Top line:

Increment byte by byte to load my name from Z register, from string 1

Store data to register, increment Y register

Decrement the counter

Loop until the counter has value 0

Z register points to string in PM

Bottom line:

Repeat steps from top line but get info from string 2.

D7:

Push wait count, inner and outer loop counts, mpr, Z register and Y register to the stack.

Load string 1 to Z register

Point Y register to top line of the LCD

Load data from first spot of line 1

Conditional check loop to see when the inner loop reaches the max value of the top line.

Marquee loop:

Push wait count and MPR to stack

Call the D5 function

Make a .25 second wait time.

Call Inverse function

Call LCD write function

Call wait function

Pop data from stack to MPR

Pop wait count from stack

Inverse function:

Push Z and Y registers to Stack, Push MPR to stack

Clear LCD

Store String 2 data in Z register

Point Y register to upper and lower DM locations

First:

Load Z register, increment

Store mpr to DM and increment

Check z to byte after the end of string 2

Keep loop going until equal.

Second:

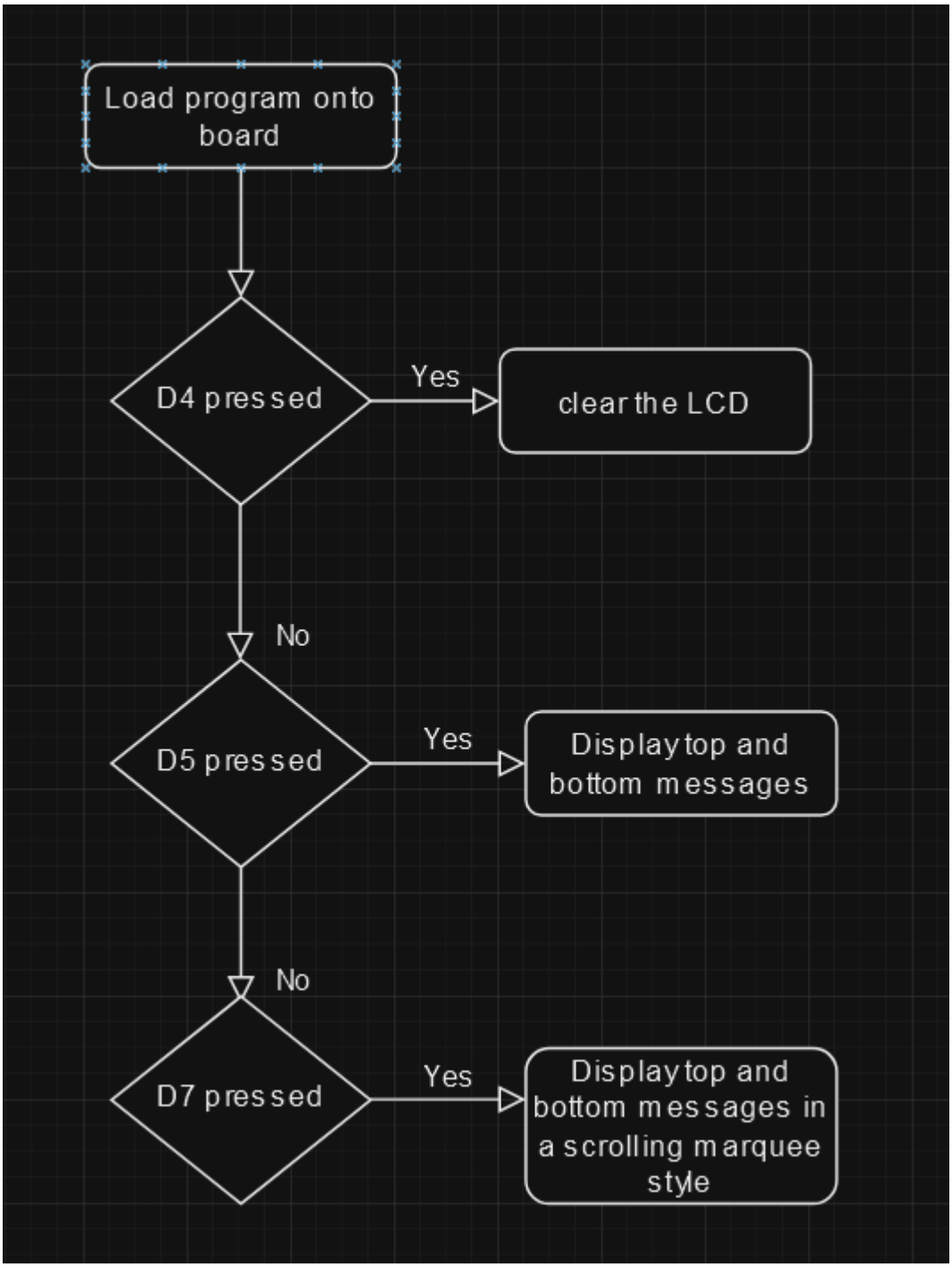
Load MPR, decrement Y

Move up 1 letter

Check if chars still need to be moved

Loop until job is done.

Pop everything back from stack to original locations



PROGRAM OVERVIEW

This program works by initializing the LCD, clearing the LCD and turning on the LCD's backlight. Once that is done, the stack pointer, Port D, DDRD, Port B and DDRB need to be initialized. The stack pointer takes the RAMEND to the MPR and puts the MPR to the SPL/SPH. Port D is initialized by loading $00 and $FF to MPR, then outputting MPR to DDRD and PORTD. The same happens with PORT B. Once those are ready to use, we call the function that sets the inputs to high, which enables them for use since they are active low. Once the buttons have been initialized, load

the PIND input to MPR. Once the input has been loaded into MPR, compare the value in MPR if D4, D5 or D7 has been pressed. From there the corresponding button function will be called. If D4 is pressed, then the LCD will be cleared of any messages it is displaying. If D5 is pressed, it will display my name, Levi Stidham, on the top line, and my message of "What up world" on the bottom line. If D7 is pressed, then the message from D5 will be displayed in a scrolling marquee style message with the top line moving to the bottom and bottom to the top as it scrolls.

INITIALIZATION ROUTINE

This starts by calling the LCDInit, LCDBacklighton, and LCDClr functions to get the LCD ready to display my message. Once the LCD is set up, the stack pointer is initialized along with PORTD and PORTB. Port D receives the input, and Port B is used for the output.

MAIN ROUTINE

This starts by calling the button initialization function which sets all the buttons to high. Then it loads the PIND input to MPR. From there it compares the MPR to 4, 5 and 7 to see if button D4,D5,D7 was pressed. From there it will execute the appropriate subroutine. These are listed below.

SUBROUTINES

1. Wait Routine

There is a wait function that is used. This function came from the source code given in Lab 1. It starts by pushing the wait count, inner loop count and outer loop count to the stack. Then the olcnt and ilcnt are loaded. Ilcnt is decremented and the inner loop continues. Decrement the outer loop and continue the outer loop. Decrement wait then continue the wait loop. Then pop all the data back from the stack.

2. D4 Routine

   There is a D4 function that is used when D4 is pressed. This simply calls the LCDClr function from the LCD driver file.

3. D5 Routine

   There is a D5 function that will display the static message stored. This starts by calling the LCDClr function, then the data is loaded from the Z register, and post incremented. Data from R16 is stored to DM and shifts by 1 in a post increment. Then the counter is decremented. This loop will continue until the counter is equal to 0. This is done for the bottom line as well.

4. D7 Routine

This routine starts by pushing waitcnt, ilcnt, olcnt, mpr, Z and Y to the stack. Z is loaded with the first string then loads the data from the first spot of line 1 and sets off a conditional check loop. Then it goes into the marqueeloop routine. The marqueloop calls the D5 routine, the inverse routine, LCDWrite routine and wait routine.

5. Inverse Routine:

This routine starts by pushing Z and Y to the stack and the MPR to the stack. Clears the LCD, and loads the string 2 to Z and loads the DM locations to the Y register. Once that is done, it loads Z to MPR and increments and stores MPR on Y and increments . Then it compares ZL to the low string and loops until they are equal. Then it moves up 1 letter by calling STD Y+, mpr, it checks if additional chars still need to be moved, and loops until it does not need anymore. It ends by popping everything back off the stack.

TESTING

Case	**Expected**	**Actual meet expected**
D4 Pressed	LCD Cleared	Yes
D5 pressed	Name and message displayed	Yes
D7 pressed	Name and message display in a scrolling marquee style	Yes
…	…	…

STUDY QUESTIONS

The difference between Program memory and Data memory is that program memory is used for instructions where Data memory is the actual data used by the program. Program memory is permanent, Data memory is temporary and used for storing intermediate results and variables.

A function call works by transferring control to the function code when it is called. The call instruction pushes the address of the instruction following the call onto the stack, the stack holds the return address for the original point, local variables and other information that only pertains to the function call. The function is executed then the RET call is made which pops the return address from the stack and gives control back to that address. Once this is done the next line from the original location is executed.

3. Add16BitNumbers:

```
; Load the low byte of num1
LD R0, $0110
; Load the high byte of num1
LD R1, $0111
; Combine the low and high bytes of num1 into a 16-bit value
LD Rd, R1:R0

; Load the low byte of num2
LD R2, $0120
; Load the high byte of num2
LD R3, $0121
; Combine the low and high bytes of num2 into a 16-bit value
LD Rd, R3:R2
```

```
    ; Add the two 16-bit numbers
    ADD Rd, R5
    ADC R1, R4 ; Add the carry from the previous addition

    ; Store the result in memory
    ST $0100, R3 ; Store the low byte of the result
    ST $0101, R4 ; Store the high byte of the result

    ; Return from the function
    RET
```

4.

```
Subtract16BitNumbers:
    ; Load the low byte of num1
    LD R0, $0110
    ; Load the high byte of num1
    LD R1, $0111
    ; Combine the low and high bytes of num1 into a 16-bit value
    LD Rd, R1:R0

    ; Load the low byte of num2
    LD R2, $0120
    ; Load the high byte of num2
    LD R3, $0121
    ; Combine the low and high bytes of num2 into a 16-bit value
    LD Rd, R3:R2

    ; Subtract num2 from num1
    SUB Rd, R5
    SBC R1, R4 ; Subtract the carry from the previous subtraction

    ; Store the result in memory
    ST $0100, R3 ; Store the low byte of the result
    ST $0101, R4 ; Store the high byte of the result

    ; Return from the function
    RET
```

DIFFICULTIES

I have not been able to get all of my buttons to function properly. Even with the extra week, I found this to be a very difficult lab. I have always kind of struggled with pointers and this lab was no exception. It is helpful to try and draw out the data movement through the functions, but I have a hard time when functions are being called from throughout the execution of the program. I think it would be helpful if we could come to the homework office hours with questions about the lab and our code. I have to work and it makes it difficult get time to make office hours on Thursday only.

CONCLUSION

This lab was a good chance to try and get more familiar with pointers and the stack. I always enjoy the labs that allow us to make something more tangible. In this case it was getting an LCD to display my name and a message. I think labs like this help to give a sense of how what we have been learning is practical and can be taken into the real world.

SOURCE CODE

```
;
;
; ECE375_Lab3.asm
;
; Created: 2/1/2024 12:01:51 PM
; Author : Levi Stidham
;


	;***********************************************************
;*
;*
;*	 Author: Levi Stidham
;*	   Date: 02/01/2024
;*
;***********************************************************

.include "m32U4def.inc"			; Include definition file

;***********************************************************
;*	Internal Register Definitions and Constants
;***********************************************************
.def	mpr = r16				; Multipurpose register is required for LCD
Driver
.def	counter = r17			;Coutner register
.def	temp = r23				; Temperary register
.def	waitcnt = r18			; wait count register
.def	ilcnt = r19				; inner loop counter
.def	olcnt = r24				; Outer Loop Counter  r14&r15 are for the
marquee style funct.
.def	inputregister = r25		; handles inputs

;***********************************************************
;*	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 LCD
		rcall LCDInit		; Initialize LCD Display
		rcall LCDBacklighton ;Turn on LCD backlight
		rcall LCDClr		;Clear the LCD


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


		;Initialize Port D for input
		ldi mpr, $00		;Initialize Port D DDRD for input
		out DDRD, mpr		;Input
		ldi mpr, $FF		;Initialize Port D DDRD
		out PORTD, mpr			;All inputs for PORTD are tri state

		;Initialize Port B for output
		ldi	mpr, $FF			;set up portB DDRB
		out DDRB, mpr		;output
		ldi mpr, $00		; Initialize Port B DDRB
		out PORTB, mpr			; places all Port B outputs low.

		; 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

		call	Button_Int		;set up buttons for inputs.

		in mpr, PIND			; Put PIND input in mpr
		cpi mpr, 0b1110_1111				; Check if a button (d4)
was pressed
		brne Check_D5
		call D4				; Go to D4 function
		rjmp MAIN			; Return to MAIN

Check_D5:
		cpi	mpr, 0b1101_1111			;Check if D5 was pressed
		brne	Check_D7
		call	D5			;Go to D5 function
		rjmp	MAIN			;return to MAIN

Check_D7:
		cpi	mpr, 0b0111_1111			;check if D7 was pressed
		brne	MAIN
		call	D7			;go to D7 function
		rjmp	MAIN			;Return to main

		call	LCDWrite		;Write message/name to 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: Button_Initialization
; Desc:       place buttons into the upper half of the mpr. Buttons will be active low
;-----------------------------------------------------------------
Button_Int:
       in           mpr, PIND                   ;Input from PIN D
       andi   mpr,0b11111111             ; clear bits
       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)-1+4)*olcnt)-1+4)*waitcnt)-1+16
;-----------------------------------------------------------------
Wait:
              push   waitcnt                        ; Save wait register
              push   ilcnt                ; Save ilcnt register
              push   olcnt                ; Save olcnt register

Loop:  ldi           olcnt, 224           ; load olcnt register
OLoop: ldi           ilcnt, 237           ; load ilcnt register
ILoop: dec           ilcnt                ; decrement ilcnt
              brne   ILoop                ; Continue Inner Loop
              dec           olcnt         ; decrement olcnt
              brne   OLoop                ; Continue Outer Loop
              dec           waitcnt               ; Decrement wait
              brne   Loop                 ; Continue Wait loop

              pop           olcnt         ; Restore olcnt register
              pop           ilcnt         ; Restore ilcnt register
              pop           waitcnt               ; Restore wait register
              ret                         ; Return from subroutine



;-----------------------------------------------------------
; Func: D4
; Desc: clear content
;
;-----------------------------------------------------------
D4:
;button D4
```

```
	rcall LCDClr		;Clear both lines


	ret



;-----------------------------------------------------------
; Func: D5
; Desc: Display name on top line, message on the bottom line
;
;-----------------------------------------------------------
;Static_Display

D5:
	rcall LCDClr                                    ;Start by clearing both lines of
the LCD of data

	 ;Initialize Y and Z registers, move strings from PM to DM
	ldi		ZL, Low(STRING_BEG01<<1)    ; Z register points to low byte of string
in PM
	ldi		ZH, High(STRING_BEG01<<1)   ; Z register points to hight bytes of
string in PM
	ldi		YL, Low($0100)                          ; Y register points to low
byte location for line 1 $0100 comes from LCDDriver
	ldi		YH, High($0100)                         ; Y register points to high
bytes location for line 1. $0100 comes from LCDDriver.
	ldi		counter, 12                             ; Load constant 16 to r23.
Using 16 characters because that is the maximum size possible. My name is only 12 chars
(including space between names)

Top_Line:

	lpm		mpr, z+                                         ;load data from Z.
Post increment to go through byte by byte.
	st		Y+, mpr                                         ;Store data from r16
to DM. Shift by 1 post increment.
	DEC		counter                                         ; decrease counter
	brne   Top_Line                                 ;Continue loop if counter isn't 0


	. ;Initialize Y and Z registers, move strings from PM to DM
	ldi		ZL, Low(STRING_BEG02<<1)    ; Z register points to low byte of string
in PM
	ldi		ZH, High(STRING_BEG02<<1)   ; Z register points to hight bytes of
string in PM
	ldi		YL, Low($0110)                          ; Y register points to low
byte location for line 1 $0110 comes from LCDDriver
	ldi		YH, High($0110)                         ; Y register points to high
bytes location for line 1. $0110 comes from LCDDriver.
	ldi		counter, 14                             ; Load constant 16 to r23.
Using 16 because it is the maximum size possible. The phrase I picked has 14 chars
Bottom_Line:

	lpm		mpr, Z+                                         ; load data from Z
register. Post increment to go byte by byte.
	st		Y+, mpr                                         ;Store data from r16
to DM. Shift by 1, post increment.
```

```
	DEC		counter					;decrease counter by
1.
	brne	Bottom_Line			;Continue loop if counter is not
yet 0.
	rcall LCDWrite					;write to lcd
	ret						;exit function


;-----------------------------------------------------------
; Func: D7
; Desc: Marquee style scrolling message
;
;-----------------------------------------------------------
D7:
	;Push a bunch of stuff to the stack
	push	waitcnt					; r12
	push	ilcnt				; r19
	push	olcnt				; r24
	push	ZH					; r31
	push	ZL					; r30
	push	YH					; r29
	push	YL					; r28
	push	mpr


	ldi XH, $01
	ldi XL, $20

	ldi counter, 32
	lop:
		ld mpr, -X
		push mpr
		dec r17
		brne lop

		ldi XL, $01
		ldi counter, 31

	lop2:
		pop mpr
		st X+, mpr
		dec r17
		brne lop2

		pop mpr
		ldi XL, $00
		st X, mpr

	ldi		ZL, Low(STRING_BEG01<<1)	;ZL to the low bits
	ldi		ZH, High(STRING_BEG01<<1)	;ZH to high bits
	ldi		YH, High($0100)
	ldi		YL, Low($0100)

	ldi		ZH, High(STRING_BEG02<<1)
	ldi		ZL, Low(STRING_BEG02<<1)	;Set Z to end of line 2
	ldi		YH, $00
	ldi		YL, $01					;Upper and lower DM
locations
```

```
	ld		mpr, Y+					;load data from
first spot of line 1
	ldi		ilcnt, low($0110)		;set up conditional check loop

	MarqueeLoop:



		ldi		waitcnt, 25			;.25s wait time
		call	LCDWrite			;Write that down Patrick! Write
that down!
		call	wait				; hold your horses. Lets take a
pause.

		pop		mpr

		pop		YL
		pop		YH
		pop		ZL
		pop		ZH					;popping the
variables from the stack.
		pop		olcnt
		pop		ilcnt
		pop		waitcnt
		ret							; Bye! Have fun back
in the main function!




;***********************************************************
;*	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_BEG01:
.DB		"Levi Stidham"		; Declaring data in ProgMem
STRING_END01:
STRING_BEG02:
.DB		"What up world "	;declaring data in ProgMem
STRING_END02:

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