ECE 375 Lab 3

Data Manipulation and the LCD

Lab session: 015
Time: 12:00-13:50

Author: Astrid Delestine
Programming partner: Lucas Plastid

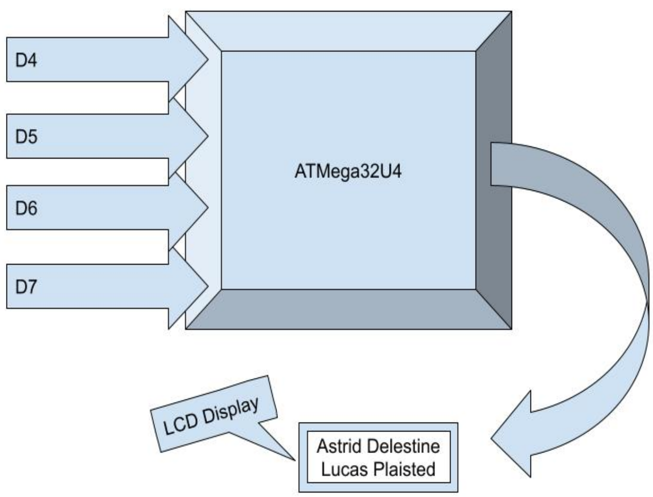
TA Signature

1 Introduction

This is the third lab in the ECE 375 series and it covers a basic introduction to the connected LCD panel, and introduces the idea of peripherals to the student. Additionally it also covers data manipulation, in the different data spaces. The students job for this lab is to use the given LCD Driver to do 3 different things, Firstly clear the display of any static or previous data, Secondly, statically print the names of the two team members to the LCD, and finally, have the LCD operate in a marquee fashion, rotating the text to the right.

2 Design

To design the program for this lab, Lucas and I brainstormed exactly what needed to happen and how we wanted it to go. We needed to determine what the buttons were going to do, and so with the main guide's and the presentation slides, we planned to have four different buttons clear the display, set the display to static text, scroll through the display in a marquee fashion and halt the marquee function when needed. Below one can see a block diagram of what our original plan was.



3 Assembly Overview

As for the Assembly program an overview can be seen below

3.1 Internal Register Definitions and Constants

The multipurpose register was setup as r16. a wait counter register was setup at r17. For the timer function an inner counter and outer counter register were setup as r18 and r19 respectively. Several different values of importance were also named such as the LCD memory locations for the first line, the second line, and the end of the second line.

3.2 Initialization Routine

Firstly the stack pointer is initialized, next the LCD display is initialized via an rcall. Finally the port D is initialized to have all inputs.

3.3 Main Routine

The main routine is quite simple, First a function is called BTN2MPR, then using the output of this function, expected to be saved to mpr, we can compare particular bits to see if buttons have been pressed. If a button, say button d5 is pressed, then an rcall is made to DISPNAMES. This will continue to loop until the end of time.

3.4 Subroutines

3.5 MARQUEE

The marquee function will shift letters from their current locations to the right and if they go off the screen they will loop around. This will happen at a stock rate of 1 movement per quarter second. This can be adjusted. Marquee will continue until button 6 is pressed. First the function loads the display with all the characters then it goes into its main loop, rotating the characters back and forth, and making sure to write to the LCD in between each moment. Some added functionality of this subroutine is that we can speed up or slow down the marquee if necessary. Once button 6 has been pressed the loop will end and the function returns to the main or wherever called it.

3.6 ROTCHAR

ROTCHAR rotates all charters right through the memory where the LCD pulls from. It does not write to the display it only edits the memory locations. It preforms this action by pushing the variables it is going to use to the stack for safe keeping, then it loads the LCD Ends into the Y pointer. It then pulls the last character and saves it into mpr and pushes it to the stack. Mpr is then loaded with the pre-decremented location of Y, causing the letter before last to be saved into mpr. it is then moved up by 1 in memory, and this will continue until the first character is moved, then the loop breaks and the first character is set to the character previously pushed to the stack. Finally all variables are popped back to their previous locations and the program counter returns to where it was before.

3.6.1 BTN2MPR

Places the 4 button inputs into the higher 4 bits of mpr. These buttons are active low. To confirm that it is only the 4 top most bits being saved into mpr an and filter is applied before returning to the main function.

3.6.2 DISPNAMES

This subroutine is quite simple, it first pushes all the variables it is going to use onto the stack. Then it loads the string locations into the Z pointer, it also loads the LCD locations into the Y pointer. Then using mpr it copies data from the Z pointer to the LCD location 1 letter at a time, until both the top and the bottom buffers have been filled. then it calls the lcd write function, and finally pops all the saved variables off the stack before returning back to the previous function.

3.6.3 Wait

The Wait subroutine controls the wait intervals while another function is preforming an action. Due to each clock cycle taking a measurable amount of time, we can calculate how many times we need to loop for. This function used the olcnt and ilcnt to have two nested loops, running the dec command until they equal zero, thus waiting the requested amount of time. **The original program was changed by modifying the Wtime constant value by shifting the bit back by 1 space inside of the HitRight subroutine and the HitLeft subroutine. This effectively doubles the wait time. See Lines 167, 201**

4 Testing

Tested each button

Case	Expected	Actual meet expected
D4 Pressed	Clears the Display	✓
D5 Pressed	Shows 2 lines of text, Each name	✓
D6 Held after D7 is Pressed	Cancels Marquee	✓
D7 Pressed	Begins Marquee	✓

Table 1: Assembly Testing Cases

5 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.

   The intended use of data memory is to store large amounts of data that is persistent on reboot. Program memory is filled on the fly and is considered more versatile. It is however cleared without power.

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.

   Function calls (rcall) are used to jump to an external to the main function, function. This allows for a cleaner, and easier to read program, that could be more efficient. One important factor of the rcall function is that the program counter is pushed to the stack when it is called, so it is very important to have the stack pointer initialized, and once a function is complete it must end with a ret call, to return to the main function, or to wherever the program counter was last.

3. Write pseudocode for an 8-bit AVR function that will take two 16-bit numbers (from data memory addresses \$0111:\$0110 and \$0121:\$0120), add them together, and then store the 16-bit result (in data memory addresses \$0101:\$0100). (Note The syntax "\$0111:\$0110" is meant to specify that the function will expect little-endian data, where the highest byte of a multi-byte value is stored in the highest address of its range of addresses.)

   ldi XH, \$01
   ldi XL, \$10
   ldi YH, \$01
   ldi YL, \$20
   ldi ZH, \$01
   ldi ZL, \$00
   CLR Z ADW Z+1:Z Y+1:Y //add word ADW Z+1:Z X+1:X // add word

4. Write pseudocode for an 8-bit AVR function that will take the 16-bit number in \$0111:\$0110, subtract it from the 16-bit number in \$0121:\$0120, and then store the 16-bit result into \$0101:\$0100

   ldi XH, \$01
   ldi XL, \$10
   ldi YH, \$01
   ldi YL, \$20
   ldi ZH, \$01
   ldi ZL, \$00
   CLR Z SUW Z+1:Z Y+1:Y //subtract word SUW Z+1:Z X+1:X //subtract word

6 Difficulties

This lab was not too difficult, however determining exactly how the marquee needed to work was definitely a challenge. After sorting out the bugs, it was extremely exciting to see text show up on the display.

7 Conclusion

This lab really helped teach just how peripherals can work, and how the ATMEGA32U4 handles certain peripherals. Several parts were challenging however these stimulating moments allowed the student to learn and understand what they needed to do at the same time.

8 Source Code

Listing 1: Assembely Bump Bot Script

```
;***********************************************************
;*  This is the skeleton file for Lab 3 of ECE 375
;*
;*   Author: Astrid Delestine & Lucas Plaisted
;*      Date: 2/3/2023
;*
;***********************************************************

.include "m32U4def.inc"            ; Include definition file


;***********************************************************
;*  Internal Register Definitions and Constants
;***********************************************************
.def    mpr = r16                  ; Multipurpose register is required for LCD Dri
.def    waitcnt = r17              ; Counting registers for wait loop
.def    ilcnt = r18
.def    olcnt = r19
.equ    lcdL1 = 0x00               ; Make LCD Data Memory locations constants
.equ    lcdH1 = 0x01
.equ    lcdL2 = 0x10               ; lcdL1 means the low part of line 1's location
.equ    lcdH2 = 0x01               ; lcdH2 means the high part of line 2's locatio
.equ    lcdENDH = 0x01             ; as it sounds, the last space in data mem
.equ    lcdENDL = 0x1F             ; for storing lcd text
;***********************************************************
;*  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 Stack Pointer
        ldi     mpr, low(RAMEND)
        out     SPL, mpr
        ldi     mpr, high(RAMEND)
```

```
        out     SPH, mpr
        ; Initialize LCD Display
        rcall LCDInit
        rcall LCDBacklightOn
        rcall LCDClr
        ; Initialize ports
        ; Initialize Port D for input (from Lab 1)
        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
        ; 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
;*  Buttons:
;*      d4: clear text
;*      d5: display names
;*      d7: NOT 6!!! marquee-style, scroll between both lines
;*          "display at the beginning of the opposite line"
;***********************************************************
MAIN:                           ; The Main program
        ; Main function design is up to you. Below is an example to brainstorm.
        rcall   BTN2MPR     ; place 4 buttons into upper half of mpr
                            ; ACTIVE LOW!!!!!!
        sbrs    mpr, 7
        rcall   MARQUEE
        sbrs    mpr, 5
        rcall   DISPNAMES
        sbrs    mpr, 4
        rcall   LCDClr

        ; Move strings from Program Memory to Data Memory

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

;-----------------------------------------------------------
; BTN2MPR: Button to MPR (BTN2MPR)
; Desc: Places the 4 button inputs into the higher 4 bits
;       of mpr. Don't forget the buttons are active low!
;-----------------------------------------------------------
BTN2MPR:
        in      mpr, PIND       ; Get input from Port D
        andi    mpr, 0b11110000 ; Clear lower 4 mpr bits
        ret


;-----------------------------------------------------------
; Func: Marquee (MARQUEE)
; Desc: Calls DISPNAMES, shifts letters (bytes) from their
;       current data memory locations to the right, and if
;       going off of the right it will enter the left of
;       the next row, waiting for .25 seconds between each
;       move. This should be carrying bytes from the low
;       address of the LCD screen and carrying them up to
;       the highest values.
;
;       I have made the executive decision to stop this by
;       pressing button
;-----------------------------------------------------------
MARQUEE:
        push    waitcnt
        push    mpr

        rcall   DISPNAMES       ; make sure the text is in mem
        ldi     waitcnt, 25     ; for 25*10ms = 250ms or .25s
mqloop:
        rcall   ROTCHAR     ; rotate characters in memory
        rcall   LCDWrite    ; write new rotation to screen
        rcall   Wait        ; wait
        rcall   BTN2MPR     ; 7:4, active low :)
        sbrs    mpr, 4      ; silly speed up
        dec     waitcnt     ; wont underflow i swear
        sbrs    mpr, 5      ; speed down >:(
        inc     waitcnt     ; overflow would take a WHILE!
        sbrc    mpr, 6      ; never used yet! EXIT BUTTON
        rjmp    mqloop      ; do it again?

        pop     mpr
```

```
        pop     waitcnt
        ret

;-----------------------------------------------------------
; Func: Rotate Characters (ROTCHAR)
; Desc: Rotates all characters right through memory
;       where the LCD pulls from, once. Relies on
;       existing data in the lcd data memory space
;       DOES NOT WRITE TO SCREEN. Only edits memory.
;-----------------------------------------------------------
/*
The example given in the lab doc:
        Line 1: _____My_Name_is_
        Line 2: _______Jane_Doe_
            delay .25s
        Line 1: ______My_Name_is
        Line 2: ________Jane_Doe
            delay .25s
        Line 1: e______My_Name_i
        Line 2: s________Jane_Do
            delay .25s

We know that the data mem locations look like this:
        Line 1: $0100 : $010F
        Line 2: $0110 : $011F
And the shift is always done to the "right", with the
shifts carrying over into the next line. Each letter is
nicely one byte, making each line 16 bytes long (why the
locations go from 0 to f as well). In terms of shifting
to the right, this means that each letter at M(x) needs
to be placed into M(x+1), except for the last letter,
which is placed into the first characters location.

As I see it, this can be done in two different ways.
    Start at the begining ($0100) and shift up

    Start at the end and shift up working backwards

Starting at the begining has the issue of needing to
hold onto the next value, as otherwise it would be
overwritten when the previous moves forwards. This
issue could maybe be overcome by using two registers
and sort of flip flopping between the two? I.e:
    Reg1 <- M(0)
    Reg2 <- M(1)

```

```
    M(1) <- Reg1 (place M(0) into M(1))
     Reg1 <- M(2)
    M(2) <- Reg2 (place M(1) into M(2))
     Reg2 <- M(3)
    M(3) <- Reg1 cycle repeats! (place M(2) into M(3))

This has the disadvantage of only working in pairs
for a nice, repeatable algorithm and in general feels
a little silly. Instead I will work backwards:

     stack <- top value
    M(top) <- M(top-1)
    M(top-1) <- M(top-2)
     ...
    M(bottom+1) <- M(bottom)
    M(bottom) <- stack
*/

ROTCHAR:
        push YH                 ; push vars to stack
        push YL
        push mpr                ; done

        ldi YH, lcdENDH
        ldi YL, lcdENDL         ; Set Y to end of line 2
        ld  mpr, Y              ; pull last character
        push mpr                ; and stack it
rotloop:
        ld mpr, -Y              ; dec Y, mpr <- m(Y)
        std Y+1, mpr            ; move letter up 1 in data mem
        cpi YL, $00             ; check if just moved first char
        brne rotloop            ; if not go again until done

        pop mpr                 ; pop last character from stack
        st Y, mpr               ; place last character at first
                                ; done with one rotation

        pop mpr                 ; pop vars from stack
        pop YL
        pop YH                  ; done
        ret

;-----------------------------------------------------------
; Func: Display Names (DISPNAMES)
; Desc: Displayes names of project members by copying from
;       data memory into program memory
```

```
;------------------------------------------------------------
DISPNAMES:
        push ZL             ; Save vars to stack
        push ZH
        push YL
        push YH
        push mpr
        push ilcnt

        ldi  ZL , low(STRING_BEG<<1)    ; Sets ZL to the low bits
                                        ; of the first string location
        ldi  ZH , high(STRING_BEG<<1)   ; Sets ZH to the first
                                        ; of the first string location
        ldi  YH , lcdH1
        ldi  YL , lcdL1
        ldi  ilcnt , 16

WINEZ1: ; While ilcnt != zero 1
        lpm  mpr, Z+
        st   Y+ , mpr
        dec  ilcnt
        brne WINEZ1

        ;z is already pointing at the second string due to how memory is stored
        ldi  YH , lcdH2
        ldi  YL , lcdL2
        ldi  ilcnt , 16

WINEZ2: ; While ilcnt != zero 2
        lpm  mpr, Z+
        st   Y+ , mpr
        dec  ilcnt
        brne WINEZ2

        rcall LCDWrite

        pop ilcnt
        pop mpr
        pop YH
        pop YL
        pop ZH
        pop ZL                          ; Pop vars off of stack

        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
;       Imported from Lab 1
;-----------------------------------------------------------------
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

;***********************************************************
;*  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     "Astrid Delestine"      ; Declaring data in ProgMem
STRING2_BEG:
.DB     " Lucas Plaisted "
STRING_END:

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

```
;*******************************************************
;*  Functions and Subroutines Template
;*******************************************************
/*
;-----------------------------------------------------------
; 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

        */
```