

ECE 375 Lab 3

Data Manipulation and the LCD

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

Author: Astrid Delestine
Programming partner: Lucas Plastid

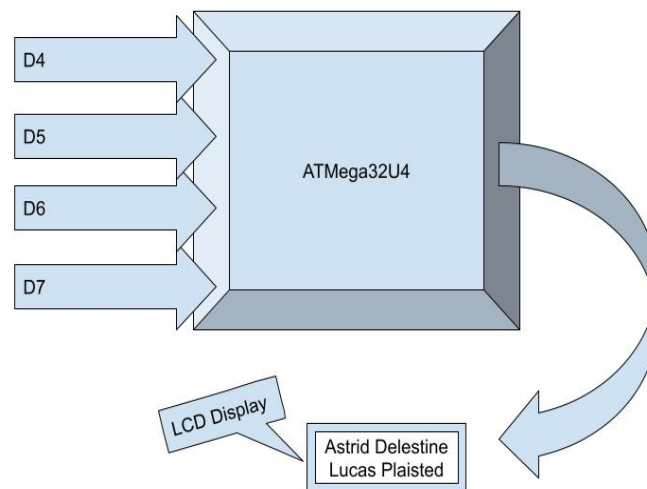
TA Signature

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 characters right through the memory where the LCD pulls from. It does not write to the display it only edits the memory locations. It performs 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 performing 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: Assembly Bump Bot Script

```

1  ;*****
2  ;*   This is the skeleton file for Lab 3 of ECE 375
3  ;*
4  ;*   Author: Astrid Delestine & Lucas Plaisted
5  ;*   Date: 2/3/2023
6  ;*
7  ;*****
8
9  .include "m32U4def.inc"           ; Include definition file
10
11
12 ;*****
13 ;*   Internal Register Definitions and Constants
14 ;*****
15 .def      mpr = r16                ; Multipurpose register is required for LCD Dri
16 .def      waitcnt = r17           ; Counting registers for wait loop
17 .def      ilcnt = r18
18 .def      olcnt = r19
19 .equ      lcdL1 = 0x00            ; Make LCD Data Memory locations constants
20 .equ      lcdH1 = 0x01
21 .equ      lcdL2 = 0x10            ; lcdL1 means the low part of line 1's location
22 .equ      lcdH2 = 0x01            ; lcdH2 means the high part of line 2's location
23 .equ      lcdENDH = 0x01          ; as it sounds, the last space in data mem
24 .equ      lcdENDL = 0x1F          ; for storing lcd text
25 ;*****
26 ;*   Start of Code Segment
27 ;*****
28 .cseg                             ; Beginning of code segment
29
30 ;*****
31 ;*   Interrupt Vectors
32 ;*****
33 .org      $0000                    ; Beginning of IVs
34          rjmp INIT                 ; Reset interrupt
35
36 .org      $0056                    ; End of Interrupt Vectors
37
38 ;*****
39 ;*   Program Initialization
40 ;*****
41 INIT:                               ; The initialization routine
42          ; Initialize Stack Pointer
43          ldi      mpr, low(RAMEND)
44          out      SPL, mpr
45          ldi      mpr, high(RAMEND)

```

```

46      out      SPH, mpr
47      ; Initialize LCD Display
48      rcall LCDInit
49      rcall LCDBacklightOn
50      rcall LCDClr
51      ; Initialize ports
52      ; Initialize Port D for input (from Lab 1)
53      ldi      mpr, $00          ; Set Port D Data Direction Register
54      out      DDRD, mpr        ; for input
55      ldi      mpr, $FF          ; Initialize Port D Data Register
56      out      PORTD, mpr       ; so all Port D inputs are Tri-State
57      ; NOTE that there is no RET or RJMP from INIT,
58      ; this is because the next instruction executed is the
59      ; first instruction of the main program
60
61      ; *****
62      ; *   Main Program
63      ; *   Buttons:
64      ; *       d4: clear text
65      ; *       d5: display names
66      ; *       d7: NOT 6!!! marquee-style, scroll between both lines
67      ; *       "display at the beginning of the opposite line"
68      ; *****
69  MAIN:                                ; The Main program
70      ; Main function design is up to you. Below is an example to brainstorm.
71      rcall    BTN2MPR          ; place 4 buttons into upper half of mpr
72      ; ACTIVE LOW!!!!!!
73      sbrs     mpr, 7
74      rcall    MARQUEE
75      sbrs     mpr, 5
76      rcall    DISPNames
77      sbrs     mpr, 4
78      rcall    LCDClr
79
80      ; Move strings from Program Memory to Data Memory
81
82      ; Display the strings on the LCD Display
83
84      rjmp     MAIN            ; jump back to main and create an infinite
85      ; while loop. Generally, every main program is an
86      ; infinite while loop, never let the main program
87      ; just run off
88
89
90
91      ; *****

```

```

92 ;*  Functions and Subroutines
93 ;*****
94
95 ;-----
96 ; BTN2MPR: Button to MPR (BTN2MPR)
97 ; Desc: Places the 4 button inputs into the higher 4 bits
98 ;       of mpr. Don't forget the buttons are active low!
99 ;-----
100 BTN2MPR:
101     in      mpr, PIND          ; Get input from Port D
102     andi    mpr, 0b11110000   ; Clear lower 4 mpr bits
103     ret
104
105
106 ;-----
107 ; Func: Marquee (MARQUEE)
108 ; Desc: Calls DISPNames, shifts letters (bytes) from their
109 ;       current data memory locations to the right, and if
110 ;       going off of the right it will enter the left of
111 ;       the next row, waiting for .25 seconds between each
112 ;       move. This should be carrying bytes from the low
113 ;       address of the LCD screen and carrying them up to
114 ;       the highest values.
115 ;
116 ;       I have made the executive decision to stop this by
117 ;       pressing button
118 ;-----
119 MARQUEE:
120     push    waitcnt
121     push    mpr
122
123     rcall   DISPNames          ; make sure the text is in mem
124     ldi     waitcnt, 25        ; for 25*10ms = 250ms or .25s
125 mqloop:
126     rcall   ROTCHAR            ; rotate characters in memory
127     rcall   LCDWrite           ; write new rotation to screen
128     rcall   Wait               ; wait
129     rcall   BTN2MPR            ; 7:4, active low :)
130     sbrs    mpr, 4             ; silly speed up
131     dec     waitcnt            ; wont underflow i swear
132     sbrs    mpr, 5             ; speed down >:(
133     inc     waitcnt            ; overflow would take a WHILE!
134     sbrc    mpr, 6             ; never used yet! EXIT BUTTON
135     rjmp    mqloop            ; do it again?
136
137     pop     mpr

```



```

138         pop        waitcnt
139         ret
140
141 ;-----
142 ; Func: Rotate Characters (ROTCHAR)
143 ; Desc: Rotates all characters right through memory
144 ; where the LCD pulls from, once. Relies on
145 ; existing data in the lcd data memory space
146 ; DOES NOT WRITE TO SCREEN. Only edits memory.
147 ;-----
148 /*
149 The example given in the lab doc:
150     Line 1: ----My_Name_is_
151     Line 2: -----Jane_Doe_
152         delay .25s
153     Line 1: -----My_Name_is
154     Line 2: -----Jane_Doe
155         delay .25s
156     Line 1: e-----My_Name_i
157     Line 2: s-----Jane_Do
158         delay .25s
159
160 We know that the data mem locations look like this:
161     Line 1: $0100 : $010F
162     Line 2: $0110 : $011F
163 And the shift is always done to the "right", with the
164 shifts carrying over into the next line. Each letter is
165 nicely one byte, making each line 16 bytes long (why the
166 locations go from 0 to f as well). In terms of shifting
167 to the right, this means that each letter at M(x) needs
168 to be placed into M(x+1), except for the last letter ,
169 which is placed into the first characters location.
170
171 As I see it , this can be done in two different ways.
172     Start at the begining ($0100) and shift up
173
174     Start at the end and shift up working backwards
175
176 Starting at the begining has the issue of needing to
177 hold onto the next value, as otherwise it would be
178 overwritten when the previous moves forwards. This
179 issue could maybe be overcome by using two registers
180 and sort of flip flopping between the two? I.e:
181     Reg1 <- M(0)
182     Reg2 <- M(1)
183

```

```

184     M(1) <- Reg1 (place M(0) into M(1))
185     Reg1 <- M(2)
186     M(2) <- Reg2 (place M(1) into M(2))
187     Reg2 <- M(3)
188     M(3) <- Reg1 cycle repeats! (place M(2) into M(3))
189
190 This has the disadvantage of only working in pairs
191 for a nice, repeatable algorithm and in general feels
192 a little silly. Instead I will work backwards:
193
194     stack <- top value
195     M(top) <- M(top-1)
196     M(top-1) <- M(top-2)
197     ...
198     M(bottom+1) <- M(bottom)
199     M(bottom) <- stack
200 */
201
202 ROTCHAR:
203     push YH                ; push vars to stack
204     push YL
205     push mpr                ; done
206
207     ldi YH, lcdENDH
208     ldi YL, lcdENDL        ; Set Y to end of line 2
209     ld mpr, Y              ; pull last character
210     push mpr                ; and stack it
211 rotloop:
212     ld mpr, -Y              ; dec Y, mpr <- m(Y)
213     std Y+1, mpr            ; move letter up 1 in data mem
214     cpi YL, $00            ; check if just moved first char
215     brne rotloop           ; if not go again until done
216
217     pop mpr                 ; pop last character from stack
218     st Y, mpr               ; place last character at first
219                             ; done with one rotation
220
221     pop mpr                 ; pop vars from stack
222     pop YL
223     pop YH                 ; done
224     ret
225
226 ;-----
227 ; Func: Display Names (DISPNAMES)
228 ; Desc: Displayes names of project members by copying from
229 ;       data memory into program memory

```

```

230 ;
231 DISPNames:
232     push ZL                ; Save vars to stack
233     push ZH
234     push YL
235     push YH
236     push mpr
237     push ilcnt
238
239     ldi ZL , low (STRING_BEG<<1)    ; Sets ZL to the low bits
240                                     ; of the first string location
241     ldi ZH , high (STRING_BEG<<1)   ; Sets ZH to the first
242                                     ; of the first string location
243     ldi YH , lcdH1
244     ldi YL , lcdL1
245     ldi ilcnt , 16
246
247 WINEZ1: ; While ilcnt != zero 1
248     lpm mpr, Z+
249     st Y+ , mpr
250     dec ilcnt
251     brne WINEZ1
252
253     ; z is already pointing at the second string due to how memory is stored
254     ldi YH , lcdH2
255     ldi YL , lcdL2
256     ldi ilcnt , 16
257
258 WINEZ2: ; While ilcnt != zero 2
259     lpm mpr, Z+
260     st Y+ , mpr
261     dec ilcnt
262     brne WINEZ2
263
264     rcall LCDWrite
265
266     pop ilcnt
267     pop mpr
268     pop YH
269     pop YL
270     pop ZH
271     pop ZL                ; Pop vars off of stack
272
273     ret
274
275 ;

```

```

276 ; Sub:   Wait
277 ; Desc:  A wait loop that is  $16 + 159975 * \text{waitcnt}$  cycles or roughly
278 ;         $\text{waitcnt} * 10\text{ms}$ . Just initialize wait for the specific amount
279 ;        of time in 10ms intervals. Here is the general equation
280 ;        for the number of clock cycles in the wait loop:
281 ;         $(((((3 * \text{ilcnt}) - 1 + 4) * \text{olcnt}) - 1 + 4) * \text{waitcnt}) - 1 + 16$ 
282 ;        Imported from Lab 1
283 ;
284 Wait:
285     push    waitcnt        ; Save wait register
286     push    ilcnt         ; Save ilcnt register
287     push    olcnt         ; Save olcnt register
288
289 Loop:   ldi    olcnt, 224    ; load olcnt register
290 OLoop:  ldi    ilcnt, 237    ; load ilcnt register
291 ILoop:  dec    ilcnt        ; decrement ilcnt
292         brne   ILoop        ; Continue Inner Loop
293         dec    olcnt        ; decrement olcnt
294         brne   OLoop        ; Continue Outer Loop
295         dec    waitcnt      ; Decrement wait
296         brne   Loop         ; Continue Wait loop
297
298     pop     olcnt          ; Restore olcnt register
299     pop     ilcnt          ; Restore ilcnt register
300     pop     waitcnt        ; Restore wait register
301     ret                    ; Return from subroutine
302
303 ; *****
304 ; *   Stored Program Data
305 ; *****
306
307 ;
308 ; An example of storing a string. Note the labels before and
309 ; after the .DB directive; these can help to access the data
310 ;
311 STRING.BEG:
312 .DB      "Astrid_Delestine"    ; Declaring data in ProgMem
313 STRING2.BEG:
314 .DB      "_Lucas_Plaisted_"
315 STRING.END:
316
317 ; *****
318 ; *   Additional Program Includes
319 ; *****
320 .include "LCDDriver.asm"        ; Include the LCD Driver
321

```

```

322 ;*****
323 ;*  Functions and Subroutines Template
324 ;*****
325 /*
326 ;-----
327 ; Func: Template function header
328 ; Desc: Cut and paste this and fill in the info at the
329 ; beginning of your functions
330 ;-----
331 FUNC:                                ; Begin a function with a label
332     ; Save variables by pushing them to the stack
333
334     ; Execute the function here
335
336     ; Restore variables by popping them from the stack,
337     ; in reverse order
338
339     ret                                ; End a function with RET
340
341     */

```